

## Controlador De Acuario Electrónico Con Arduino

Hola:

Pretendo describir en este post como he construido una especie de pequeña computadora para controlar un acuario de 270L de agua dulce, sobre este tema hay muchísima información en internet y todos ellos están realizados con una pequeñísima placa, con un microprocesador llamada Arduino (es una placa de licencia libre), las hay de varios tipos y de varios precios, la más común es la Arduino UNO que ronda los 20€ yo e elegido en cambio la Arduino Mega que ronda los 46€ pero porque no me quería quedar corto y quería que pudiera ser en un futuro ampliable el controlador.

A primera vista puede parecer que este tipo de montaje sea algo demasiado complicado para alguien que no sepa de electrónica y programación, pero les puedo asegurar que son muchas las personas que sin estos conocimientos están montando controladores de este tipo unos menos complejos y otros más pero todos se guían por lo que ven montar a los demás y consultar en foros del tema.

### [Arduino MEGA 2560 ATMEGA2560 AVR USB Board 5V + USB Cable](#)

Teniendo en cuenta que lo que hay comercializado como ordenador, son cosas como el Profilux, que solo la unidad central cuesta unos 600€, merece la pena mirárselo.

Otra cosa que hay que tener en cuenta es que los precios que he dicho son de artículos comprados en España, yo en cambio los e comprado en Ebay por menos de la mitad.

Bueno el controlador que e montado tiene las siguientes características:

Control de la temperatura del acuario y sus calentadores.

Control de la temperatura de la tapa del acuario y los ventiladores de esta.

Control de la temperatura del depósito y su calentador para los cambios de agua.

Control del PH y la electro-válvula del CO2.

Control del llenado de agua de ósmosis o agua de lluvia del depósito, con alarma de nivel que avisa y desconecta el llenado.

Control del llenado del acuario desde el deposito.

Control de las luces LEDs dimeando las luces a su hora de encendido y apagado, creado de esta forma el amanecer y el anochecer, controlado dos fases de luces LEDs de 84W = 168W lo que equivales a mucho mas de 360W de luz de fluorescentes. Y una tercera fase que controla la luz de luna.

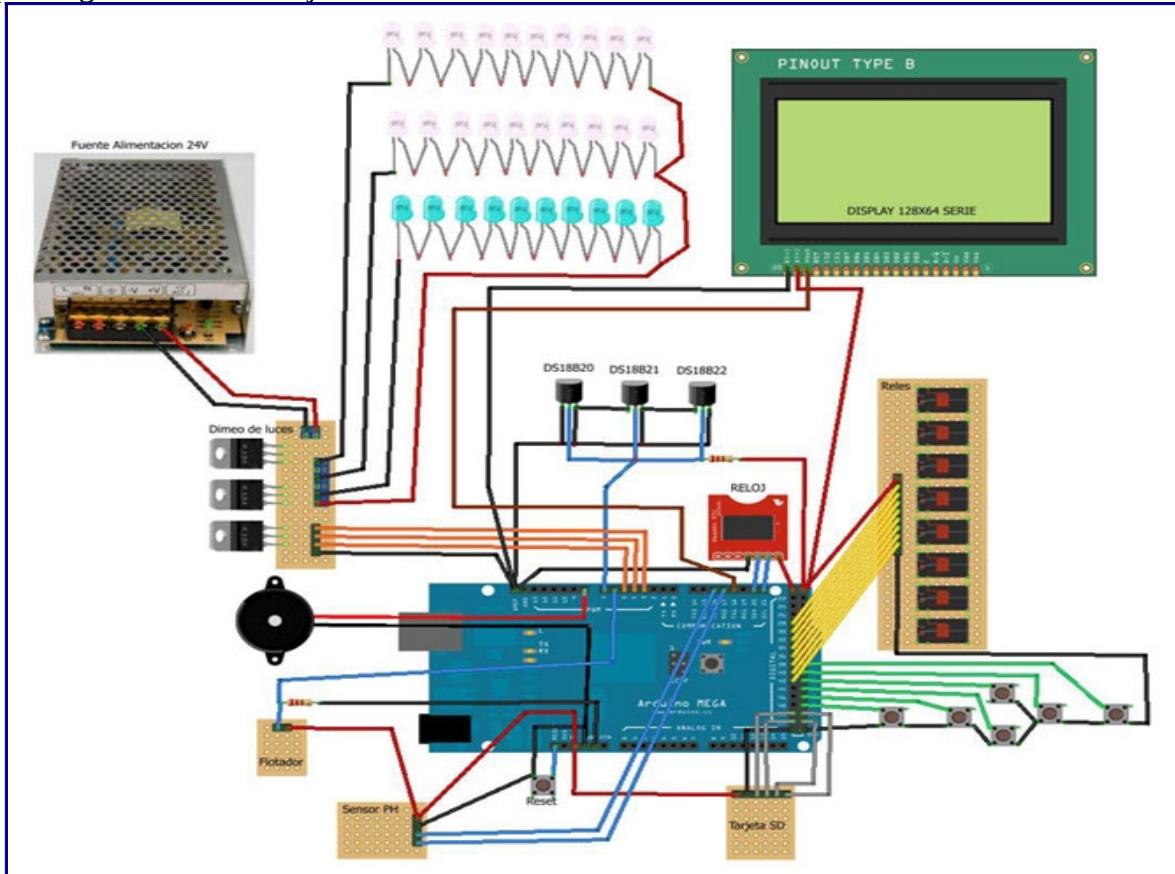
Además lleva una tarjeta SD en la cual me va grabando todos los eventos que se producen, tanto encendidos, apagados y errores, para poder luego sacar estadísticas de ello.

Este controlador lo tengo montado con este acuario que detallo [aquí](#)

Se puede hacer mucho más sencillo, por ejemplo que controle solo la luz y la temperatura del acuario, pero a mí me dio por querer complicarme la vida, os puedo decir también que se un poco de electrónica y soy programador, pero hace 3 meses no sabía la existencia de estos dispositivo y hoy tengo uno funcionando y de maravilla.

Os voy a poner un enlace a los distintos tipo de Arduino que hay, que se me había olvidado jejeje.  
<http://arduino.cc/en/Main/Hardware>

El esquema general del montaje es este



y el frontal del controlador a quedado así:



Si os interesa sigo con la descripción detallada de todo el montaje y programación.

Bueno empecemos, el controlador está basado en un microprocesador muy popular y barato el cual es el cerebro de todo el controlador, este tiene conexiones digitales y analógicas, así como conexiones SPI y serie.

Para los neófitos del tema:

**Las conexiones digitales** son aquellas que solo funcionan mandando o recibiendo una señal de SI o NO, siendo el NO que la conexión tenga menos de 2v y el sí que la conexión tenga 5v, con esto podemos gestionar salidas que manejen un Relé, un transistor de potencia o un led, si la conexión es de entrada podemos gestionar, un pulsador, un sensor de varios tipos como flotadores o sensores de presión y algunas cosas más, el que sea la conexión de salida o entrada lo definimos nosotros,

**Las conexiones analógicas** son las encargadas de manejar señales en las cuales detectamos variaciones

de tensión o generamos variaciones de tensión, dependiendo si las configuramos como entrada o salida, un uso puede ser manejar un motor, un sensor de temperatura, entre estas hay un pequeño grupo que son PWM (Pulse Wave Modulation, modulación de onda por pulsos) las cuales son para controlar una salida escalonada desde 0 a 255, las cuales son perfectas para manejar un flujo de corriente como puede ser el dímeo(encendido escalonado) de las luces de nuestro acuario.

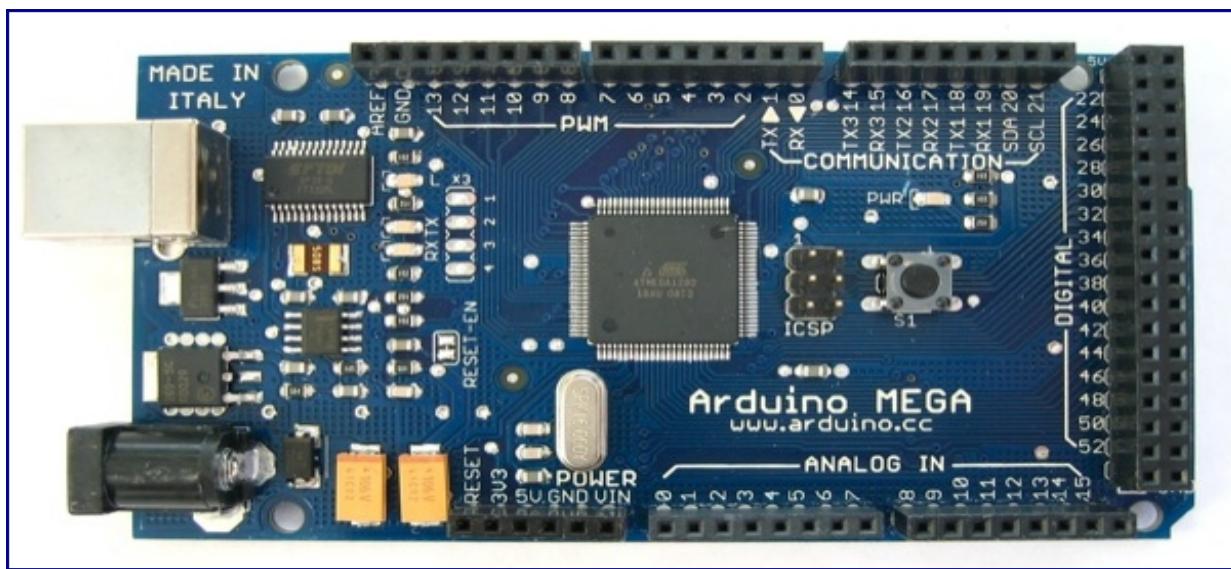
**SPI** ( Serial Peripheral Interface) es un protocolo de comunicaciones que va por un bus de tres líneas, sobre el cual se transmiten paquetes de información de 8 bits. Cada una de estas tres líneas porta la información entre los diferentes dispositivos conectados al bus. Cada dispositivo conectado al bus puede actuar como transmisor y receptor al mismo tiempo, por lo que este tipo de comunicación serial es full duplex. Dos de estas líneas trasfieren los datos (una en cada dirección) y la tercer línea es la del reloj. Algunos dispositivos solo pueden ser transmisores y otros solo receptores, generalmente un dispositivo que trámite datos también puede recibir.

En mi caso yo manejo la tarjeta SD de 2Gb para leer y escribir.

**Puertos Serie** estos son unos pines para la comunicación serie (RS-232) equivalentes al puerto serie de PC, pero que en vez de un conector de 9 puntas como el PC son de 2 conexiones (TX) trasmisión y (RX) recepción, el número de puertos serie depende del modelo Arduino, lo normal es 1 menos el Mega que tiene 4, hay que tener en cuenta que los Arduino la mayoría viene con un Puerto USB para la programación y comunicación con el, en el caso de los que solo tienen 1 puerto serie, este es el mismo que utiliza el USB.

En mi montaje los puertos serie se utilizan para comunicarse con la pantalla LCD y el sensor de PH.

El Arduino Mega que es el utilizado para este controlador es este:



Micro-controlador ATmega1280

Voltaje de funcionamiento 5V

Voltaje de entrada (recomendado) 7-12V

Voltaje de entrada (límite) 6-20V

Pines E/S digitales 54 (14 proporcionan salida PWM)

Pines de entrada analógica 16

Intensidad por pin 40 mA

Intensidad en pin 3.3V 50 mA

Memoria Flash 128 KB de las cuales 4 KB las usa el gestor de arranque(bootloader)

SRAM 8 KB

EEPROM 4 KB

Velocidad de reloj 16 MHz

## Alimentación

El Arduino Mega puede ser alimentado vía la conexión USB o con una fuente de alimentación externa. El origen de la alimentación se selecciona automáticamente.

Las fuentes de alimentación externas (no-USB) pueden ser tanto un transformador o una batería. El transformador se puede conectar usando un conector macho de 2.1mm con centro positivo en el conector hembra de la placa. Los cables de la batería puede conectarse a los pines Gnd y Vin en los conectores de alimentación (POWER)

La placa puede trabajar con una alimentación externa de entre 6 a 20 voltios. Si el voltaje suministrado es inferior a 7V el pin de 5V puede proporcionar menos de 5 Voltios y la placa puede volverse inestable, si se usan mas de 12V los reguladores de voltaje se pueden sobrecalentar y dañar la placa. El rango recomendado es de 7 a 12 voltios.

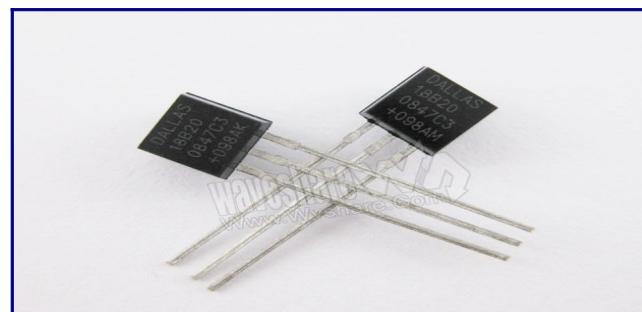
Seguiré explicando los accesorios que se le puede conectar al Arduino.

## Sensor de temperatura

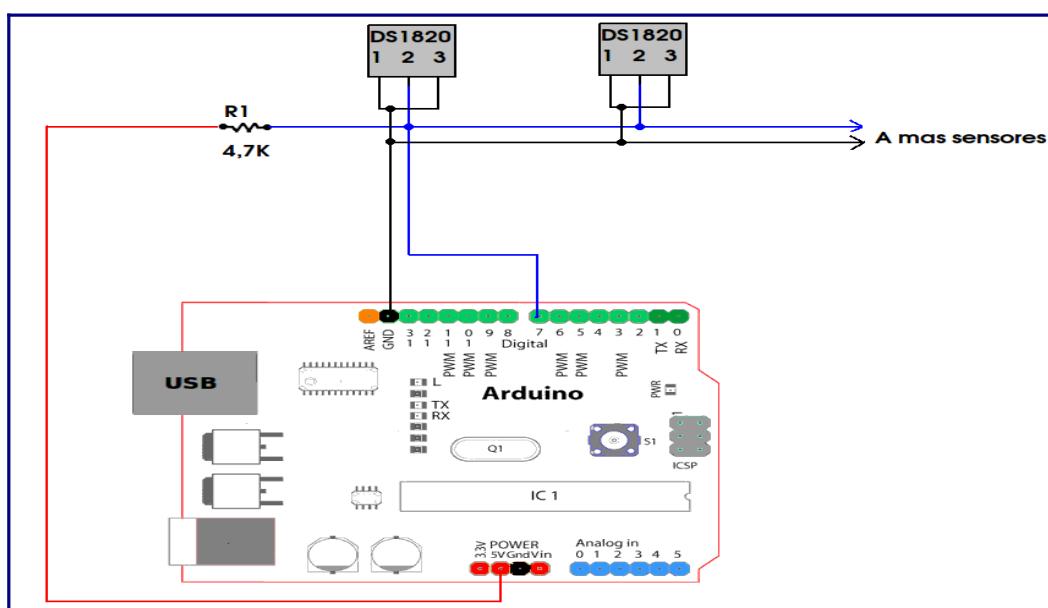
Como sensor de temperatura hay muchos modelos, el mas simple son las resistencias NTC o PTC, son unas resistencias que varían su valor según la temperatura y su lectura es analógica y cada sonda ocuparía una entrada analógica, pero el más fiable para obtener una lectura digital de temperatura es el integrado DS18B20 de la gente de Dallas.

El DS18B20 es otra maravilla de integrado, tiene un tamaño minúsculo y solamente tres patas.

No necesita alimentación exclusiva, se puede alimentar en modo parásito por la misma línea de datos.



Para comunicarse con el Arduino utiliza el protocolo 1-wire utilizando una sola entrada digital de éste (en mi caso la 7, PWM) y se utiliza esa única entrada para todas las sondas que quieras poner en línea. Para alimentarlo en modo parásito hay que poner una resistencia de 4.7K a 5Vcc (lo que se llama un pull-up) en esa misma entrada digital.



Para poder usar el DS18B20 como sonda hay que hacer un pequeño trabajo con ella, soldándole un cable y encapsulándola en un tubito sellado de silicona para que no le afecte el agua, yo compe un integrado (7€) para probarlo y cuando ya tenía la sonda echada con bastante trabajo y poca satisfacción del resultado, me dieron un enlace de Ebay para comprarla ya echada la sonda y además a 4€, con 3 m de cable, por supuesto compe unas cuantas.

En mi controlador e instalado 3, y muy fácil de conectar y programar, ya que se conectan todas en paralelo.

### [3m Digital DS18B20 Temp probe thermometer Rohs](#)

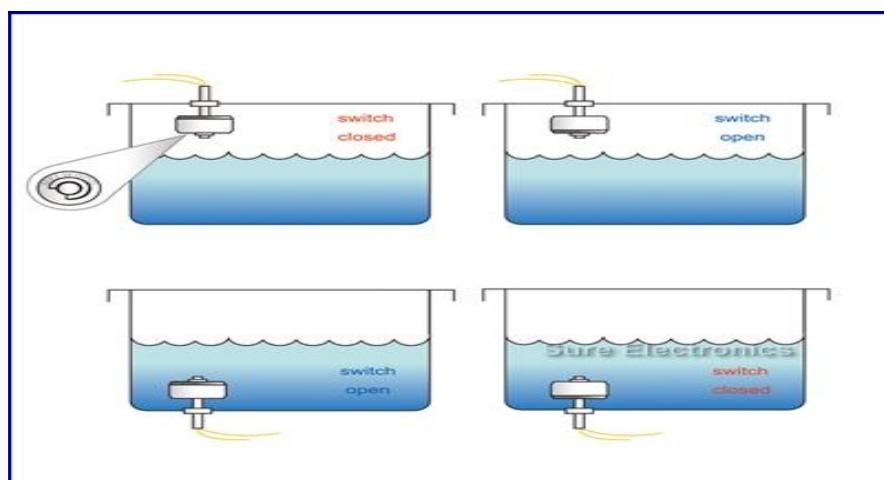
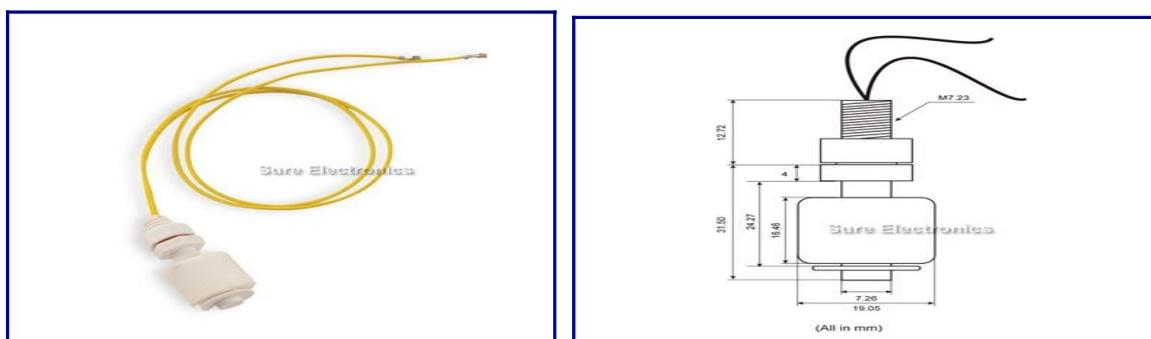
Aquí tenéis algunas imágenes del sensor



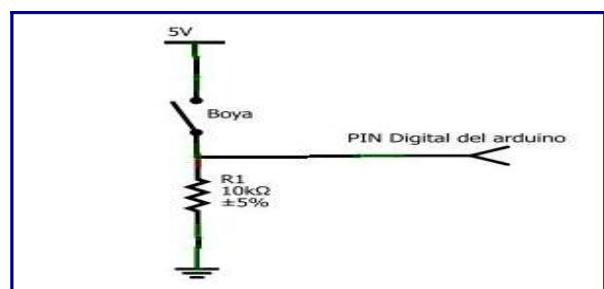
### **Sensor de nivel (Boya)**

El sensor de nivel (una simple bolla) es de lo más simple, de echo es simplemente un interruptor que se acciona por una boya que se mueve al contacto con el agua.

### [Liquid Water Level Sensor Horizontal Float Switch](#)



El cual se conecta a una entrada digital del Arduino, desde la cual podemos saber si esta activa o no.



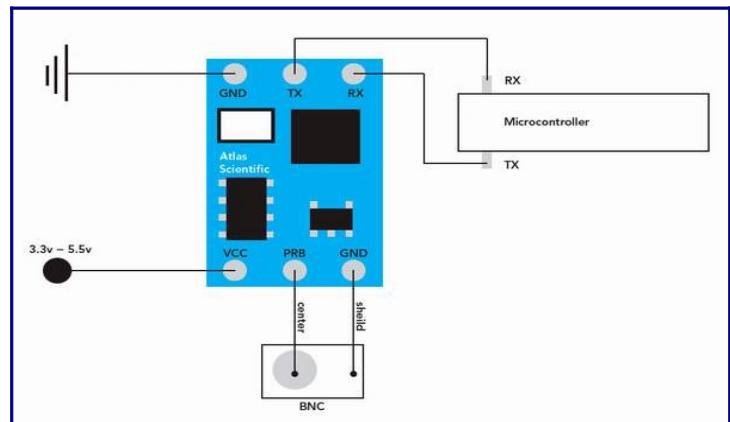
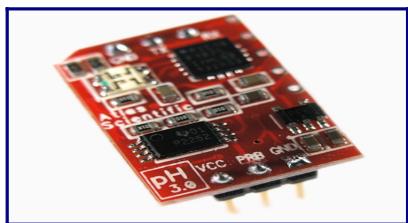
## Sensor de PH

Ya sé que el PH siempre se ha utilizado con aparatos electrónicos bastante caro, pero con el Arduino esto es mucho más sencillo, en el mercado (sobre todo por internet) hay muchos sensores de PH y de otras más cosas como de ORP, EC y de oxígenos disuelto en el agua.

En el caso de mi montaje, yo solo puse un sensor de PH y de los modelos que hay me decidí por uno de Atlas-Scientific el PH Stamp con unas características muy altas y lo principal de fácil ajuste, el coste 30€ con los gastos de envío desde USA.

[Pagina del sensor](#)

[Documentación](#)



Se conecta al Arduino por uno de sus puertos serie, por lo que es lo mismo 2 pines uno de TX y otro de RX.

La sonda de cristal que usa el sensor y que conocerán de otros aparatos de control de PH se puede conseguir barata también por Ebay.

Esta por ejemplo viene con dos botes de tapo de PH 4 y PH 7 para poder ajustarla



<http://rover.ebay.com/rover/0/e1100...BSA%3AUS%3A1123>

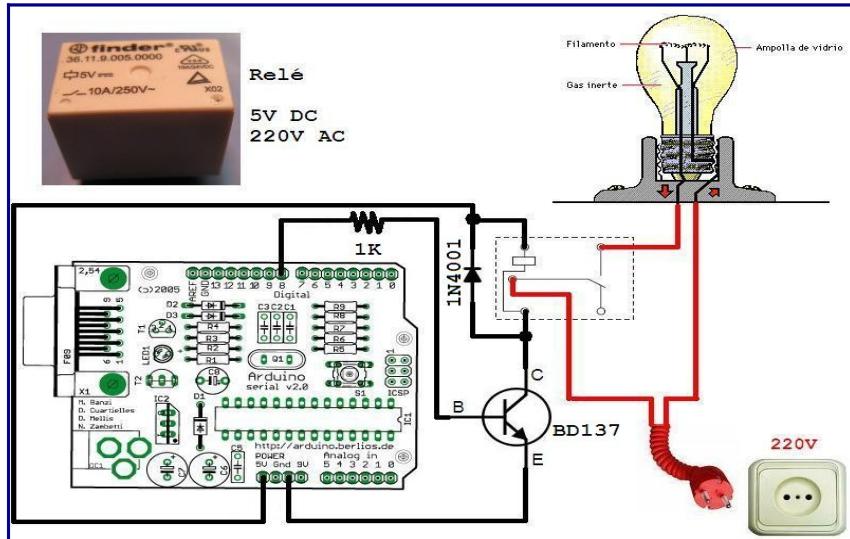
Los dos botes de tapón son importantes, pues lo necesitamos para ajustar la sonda, un tapón de PH es un líquido que tiene un PH fijo el cual nos sirve para ajustar el sensor.

Este sensor es muy simple el ajuste, basta con meter la sonda en el bote de PH 4 y mandarle una orden a través del Arduino, para que el Sensor se ajuste él solo, lo mismo con el PH7.

Con este sensor podemos desde el Arduino controlar y mostrar el PH y actuar en consecuencias, por ejemplo abrir o cerrar la electro válvula del CO2.

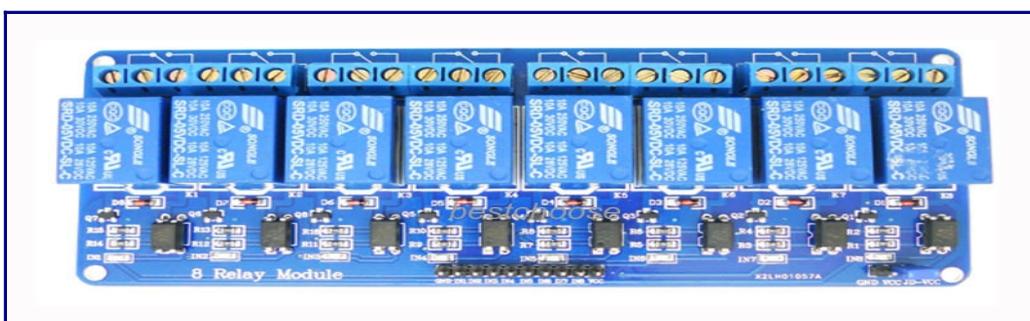
## Los relés

Los relés es una parte importante de nuestro sistema ya que son los encargados de encender y apagar los aparatos de nuestro acuario, para poder controlar desde el Arduino un relé es necesario un circuito amplificador que consta de una resistencia y un transistor, además de un diodo de protección.

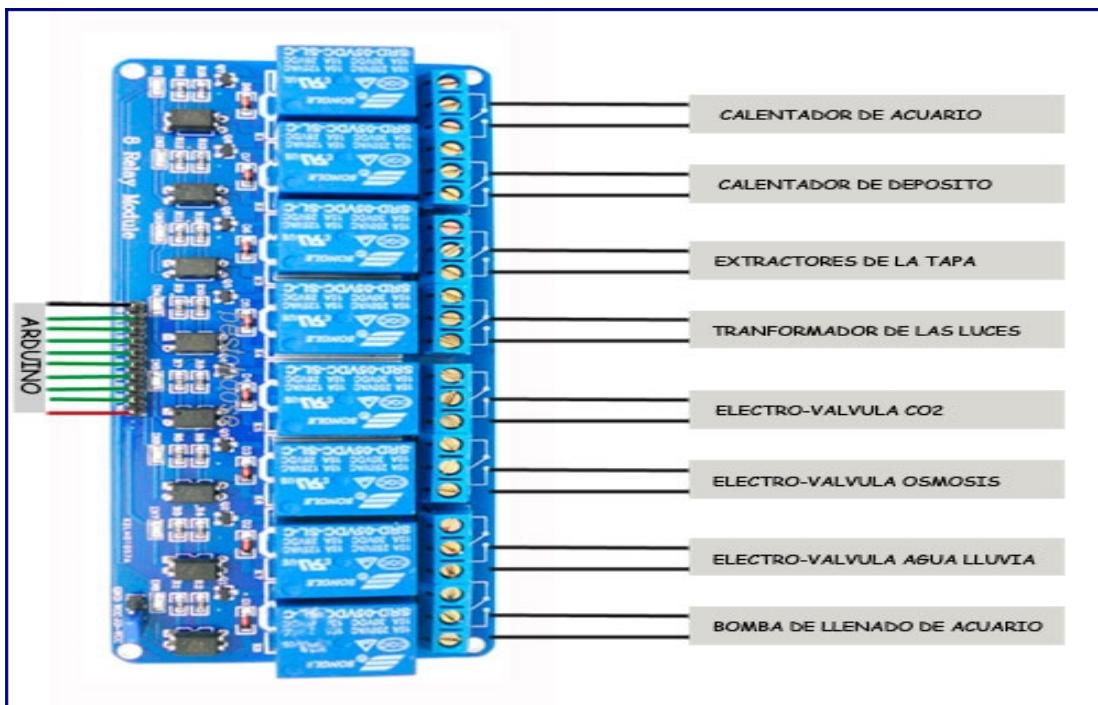


Pero actualmente es poco rentable montarnos una placa de relés ya que solo un relé nos puede costar 3€ o 4€, en cambio en Ebay tenemos placas ya montadas de por ejemplo 4 relés a 5€ con gastos de envío y todo.

Podemos encontrar placas de relés de 4, 8 y 16 relés, yo me decidí para mi montaje por una de 8 relés.



La placa la conecte de la siguiente forma.



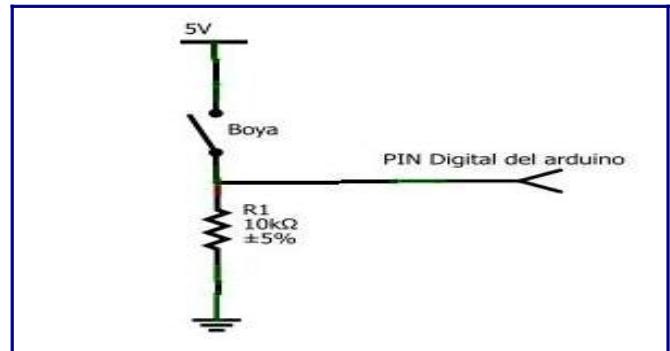
Tenéis que tener en cuenta que yo estoy describiendo el controlador que yo he creado y con los periféricos que yo he decidido, vosotros tenéis que decidir que deseáis controlar y que accesorios usar, con lo cual puede ser más pequeño o más grande.

## Teclado

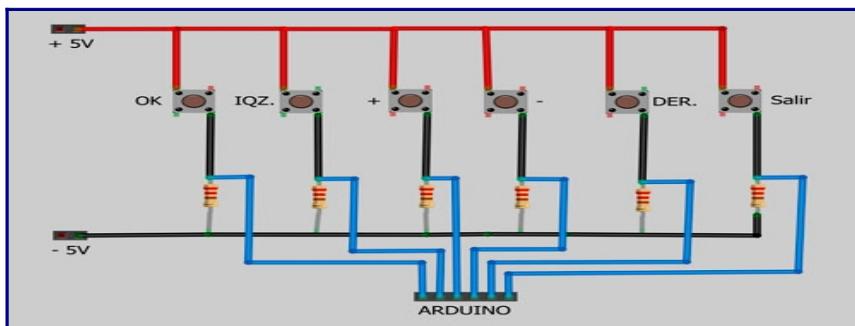
El teclado es una parte importante del controlador ya que es con lo que nos comunicamos con el Arduino, de la misma forma que nos comunicamos con los PC, por ejemplo para configurar a qué temperatura queremos el acuario o a qué hora se inicia el encendido de las luces.

He visto montajes del Arduino con tan solo 2 pulsadores como teclado y otros con un teclado de 16 teclas, todo depende de lo que queramos hacer, hay que tener en cuenta que si ponemos menos pulsadores que movimientos, tendremos que programar mas.

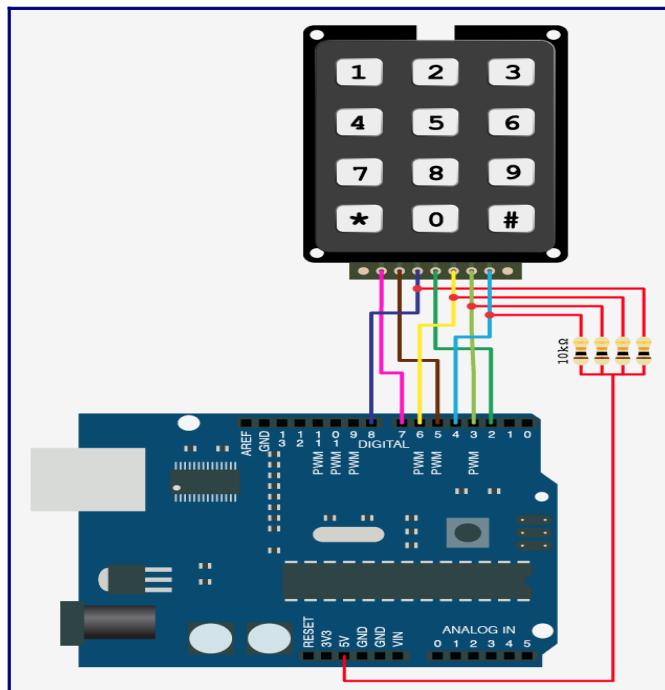
Para accionar un pulsador o tecla con el Arduino, es igual que la conexión de la bolla, la bolla lo que hace es cerrar un contacto al accionarse, lo mismo que nosotros al pulsar un pulsador.



De esta forma podemos decidir cuantos pulsadores o teclas queremos en nuestro montaje, en mi caso me decidí por 6 pulsadores, uno para el OK, otro para salir, 2 para el más y menos, para aumentar o disminuir los valores y 2 para derecha e izquierda que me servía para moverme por la pantalla.



También podéis decidiros por un teclado de los muchos que hay, de 6 a 16 teclas, suelen ser más fácil de programar.



## La pantalla

La pantalla es la parte del montaje en la que veremos el estado e información del controlador, además de servirnos para ver las operaciones que queramos hacer con el controlador, por ejemplo la configuración de los parámetros de funcionamiento como a qué temperatura queremos el acuario o a qué hora se enciende y apagan las luces.

Lo normal es que la pantalla sea una LCD de cristal líquido, las más comunes son de caracteres y se definen por cuantos caracteres caben en cada línea y cuantas líneas tiene.

<http://www.ebay.es/item/250898138727...984.m1497.l2649>



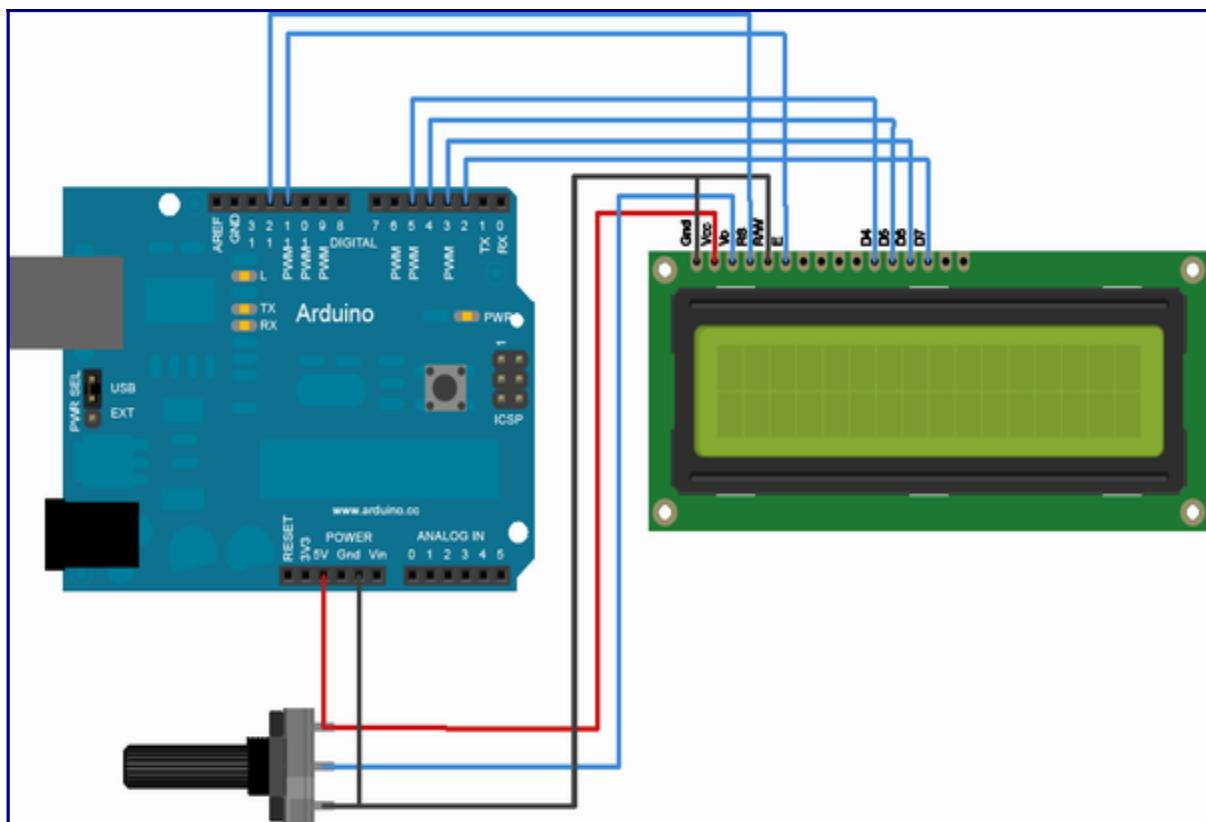
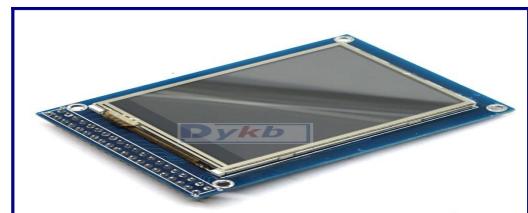
También las hay gráficas las cuales se definen por puntos (píxeles) de resolución de la pantalla, como por ejemplo la que yo utilizo en mi montaje, una de 128x64 puntos, teniendo en cuenta que cada carácter ocupa 8x6 puntos esto nos da unos 19 caracteres por línea y 8 líneas, y también tiene la posibilidad de pintar líneas, círculo, rectángulos y figuras.

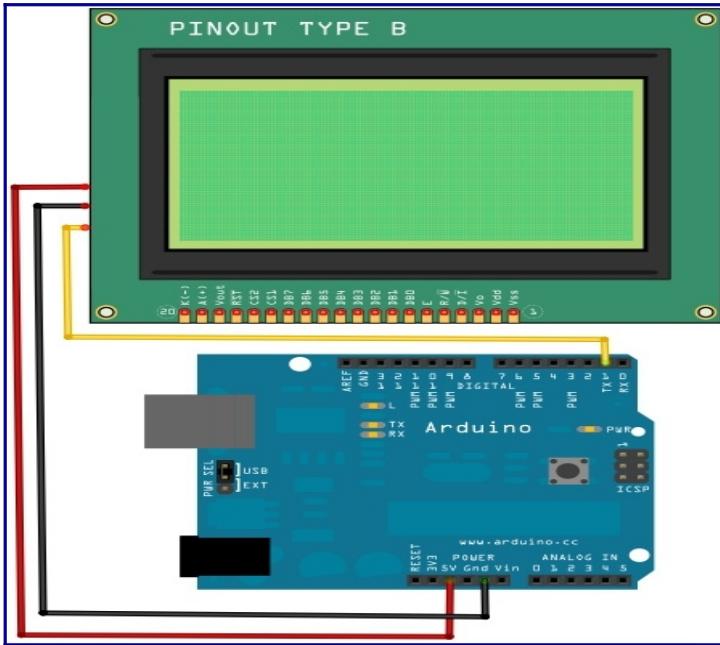
<http://www.ebay.es/item/LCD12864-LCD...=item3a6ab1c64f>



También hay otro tipos de pantallas más complejas como las gráficas a todo color y las táctiles, pero estas son más caras y más complejas, de echo la gente usa hasta las pantallas de los móviles, pero esto nos hace falta tener bastante más experiencia para usar esta opciones.

<http://www.ebay.es/item/170709357972...984.m1497.l2649>





Las pantallas son el elemento que más conexiones utilizan ya que para mandar la información a una de esta pantalla suele rondar por 8 conexiones más la alimentación.

Hay otro sistema de conexión que se utiliza, aunque también ralentiza un poco el pintado de la pantalla, y es un modulo que tiene algunas pantallas de comunicación serie, estos módulos nos permite manejar la pantalla con tan solo una conexión o dos, mas la alimentación, encargándose el modulo de convertir las ordenes del serie a ordenes de 8 hilos.

Estas pantallas las veremos que en la definición de su nombre le incluyen la denominación de “serie”.

## Lectora de tarjeta de memoria SD

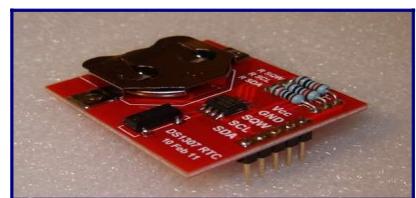
Entre las miles de cosas que podemos conectar a un Arduino, está la lectora de memoria SD, este componente nos permite leer y escribir en una tarjeta SD la información que queramos.



Este componente es raro que os haga falta ya que es muy específico y yo lo utilizo aprovechando mis conocimientos de programación para que el Arduino baya escribiendo en unos archivos de texto todo los eventos y errores que ocurran durante todo el funcionamiento del Arduino, para luego pasarlo al ordenador a través de un programa echo por mi el cual me saca estadísticas del funcionamiento del acuario, de esta forma puedo saber por ejemplo en que momentos del día se activan mas los calentadores que tiempo total al día están funcionando y algunas cosas mas.

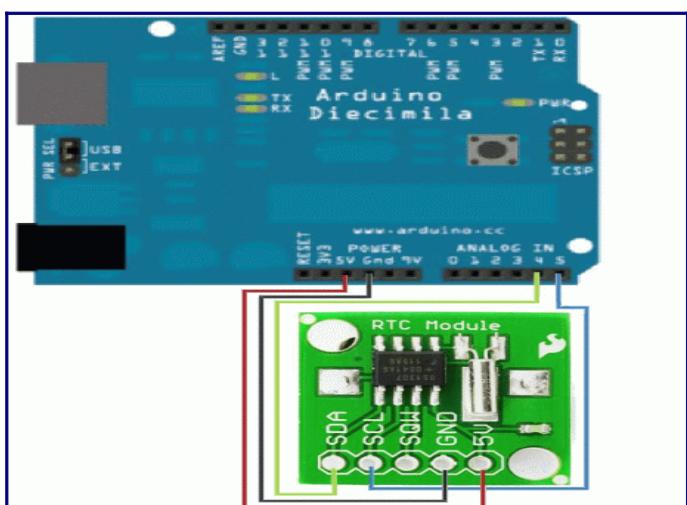
## El reloj de tiempo real.

Y por ultimo veremos el modulo de reloj, este modulo es en si un reloj el cual lleva una pequeña pila de forma que una vez puesto en hora aunque le cortemos la fuente de alimentación el mantendrá siempre la hora actual, este modulo una vez puesto en hora solo nos servirá para



preguntarle en cualquier momento que fecha es o que hora es, algo esencial para nuestro controlador, por ejemplo para saber a que hora encender las luces o apagarlas.

<http://www.ebay.es/itm/DS1307-RTC-M...=item19cb440f8e>



El modulo de reloj se comunica con el Arduino por dos conexiones a través del protocolo I2C, un sistema de comunicación soportado por algunas de las conexiones del Arduino.

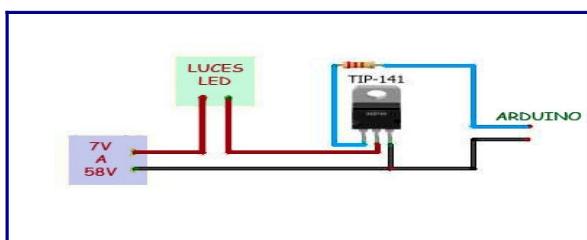
## El control de luces

Se me olvidaba un componente, el mas importante de todos el control de las luces, sobre este tema se a hablado mucho por internet, el sistema que mas utilizan la mayoría es unos transformadores dímeables que por cierto son bastante caro y para dímejar estos utilizan circuitos con un integrado regulador que le da una señal a los transformadores de 0v a 10v.

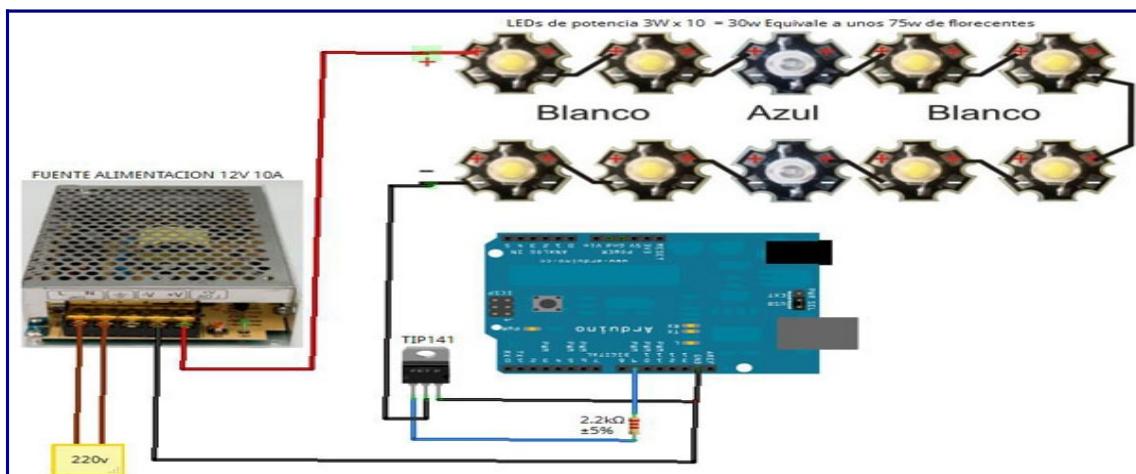
Para mi siempre e visto esto como algo demasiado complicado y caro, por lo que me decidí por un sistema mas directo de dímeo, utilizando una fuente de alimentación normal para leds y dímeando en su salida con un solo transistor.



El sistema es simple partiendo de un transformador específico para leds como este.



Obtenemos los 12v o 24v para alimentar nuestros leds, y para dímejarlos intercalamos un transistor de potencia entre este transformador y los leds, encargándose este de regular el voltaje que le llegan a los leds, de esta forma tenemos el efecto tan deseado para nuestro acuario de amanecer y anochecer.



En mi montaje yo uso 3 transistores porque he dividido las luces en 3 fases 2 en los led's de iluminación y la tercera para la luz de luna.



## **Programación del Arduino**

Bueno una vez que conocemos la mayoría de los periféricos que puede llevar nuestro Arduino, veremos como programarlo para que haga todo lo que deseemos, NO, no podemos programarlo para que haga dinero pero si para que nos controle nuestro acuario como nosotros deseemos.

Para programar el Arduino de utiliza un programa que es como un editor de texto, pero con algunas opciones específica, denominado como IDE de programación el cual es gratuito, lo hay para Windows, Mac y Linux, os lo podéis descargar de [aquí](#).

Un vez descargado el archivo comprimido lo descomprimimos en un directorio y ya lo tenemos listo para usar.

Ya solo nos falta instalar el driver del Arduino para poder manejarlo por USB, para ello basta con conectar el Arduino al PC con su cable USB y el PC nos pedirá que si busca el driver, le diremos que no que lo que queremos es que lo busque en una carpeta determinada, y esta es el subdirectorio del IDE de Arduino denominado “drivers”, con esto nos instalara el driver y tendremos todo listo para empezar.

Pensaba escribir un gran tocho describiendo los primeros paso de programación y de como utilizar el IDE, pero es algo ya esta demasiado escrito por todas parte y mejor descrito de lo que lo podría hacer yo, por lo que mejor os dejo unos enlaces a unos tutoriales que lo explican todo mucho mejor, a sin además los podéis tener para consultas, además de otros enlaces importantes.

Página oficial de Arduino, hay documentación, ejemplos y librerías para poder manejar todo tipo de periféricos, además de una guía detallada del lenguaje de programación.(en inglés)

<http://arduino.cc/>

Página oficial de Arduino en Español con lo mismo que la de inglés pero menos actualizado.

<http://arduino.cc/es/>

Página de descarga del IDE del Arduino.

<http://arduino.cc/es/Main/Software>

Página del foro del Arduino, tiene un apartado en español, bastante activo.

<http://arduino.cc/forum/index.php>

Varios tutoriales del IDE de Arduino y de programación del Arduino.

[Arduino\\_user\\_manual\\_es.pdf](#)

[Manual\\_Programacion\\_Arduino.pdf](#)

[Arduino\\_programing\\_notebook\\_ES.pdf](#)

[Arduino\\_Circuitos\\_básicos\\_de\\_entrada-salida.pdf](#)

[ArduinoTutorial.pdf](#)

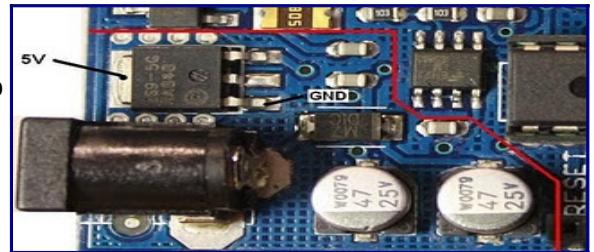
[libreto.pdf](#)

## **La alimentación de Arduino**

Antes de empezar a conectar cosas al Arduino, tenemos que tener en cuenta las limitaciones que tenemos según la forma de alimentar el Arduino, como ya dijimos lo podemos alimentar a través del cable de USB con el cual tendremos poca potencia porque esta conexión nos dará unos 250 mA, teniendo en cuenta que ya la placa del Arduino consume unos 50 mA y que cada pin de el soporta un consumo de 40 mA con lo que con pocas cosas que conectemos nos quedaremos sin potencia.

Si lo conectamos por el conector de alimentación externa tendremos que alimentarlo con un voltaje entre 6v y 12v, esto es gracias a un integrado regulador de 5v que tiene instalado el Arduino, el cual convierte

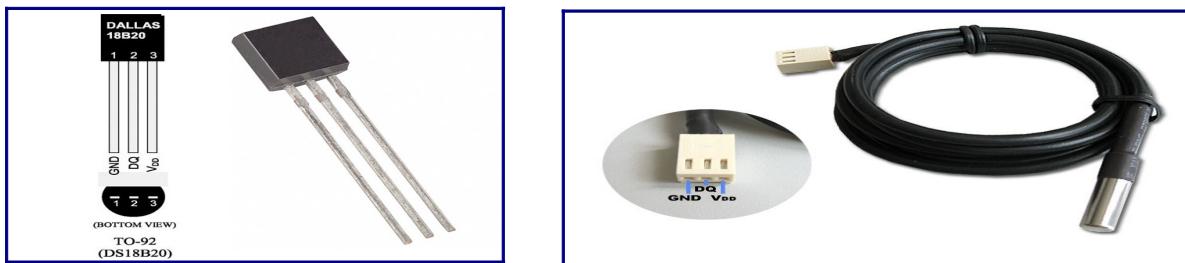
el voltaje de entrada a una salida constante de 5v, claro que el voltaje que sobra el integrado lo tiene que disipar en forma de calor, además del calor que le produce el consumo que pasa por él, con lo cual tenemos que a la larga nos permite alimentarlo con otro voltaje pero con casi el mismo consumo que el USB si no queremos quemar el integrado de entrada.



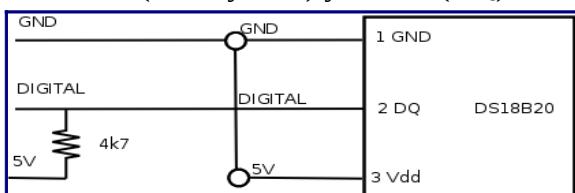
Lo más conveniente si conectamos muchas cosas es alimentarlo directamente a 5v por el conector POWER, de esta forma no tendremos problemas de consumo y podremos conectar todo lo que queramos, pero siempre que uno de los aparatos conectados necesite alimentación se conecte esta a los 5v de la alimentación directamente, además hay que tener en cuenta que el Arduino solo soporta un consumo interno de 400 mA, por lo que todas las señales de salida juntas del Arduino no tienen que superar ese valor.

### Conectando nuestro primer artilugio al Arduino (La sonda de temperatura)

Como ya vimos al principio del post, como sonda de temperatura lo mejor que hay es el integrado de 3 patillas DS18B20 de la casa Dallas/Maxim es un termómetro digital de 9 a 12 bits que se conecta a través de un bus 1-wire, mide la temperatura en grados Celsius, tiene una operación de temperatura que va del rango de -55°C a +125°C.

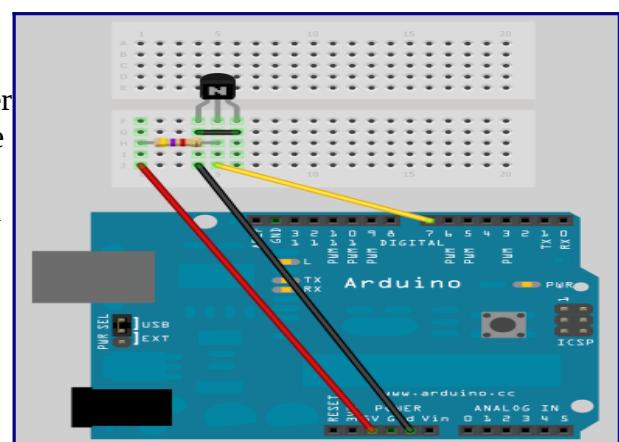


El DS18B20 es un circuito integrado con forma de transistor, tiene tres pines de los cuales dos sirven de alimentación (GND y Vdd) y el otro (DQ) es el que se conecta al bus. Este sensor puede funcionar

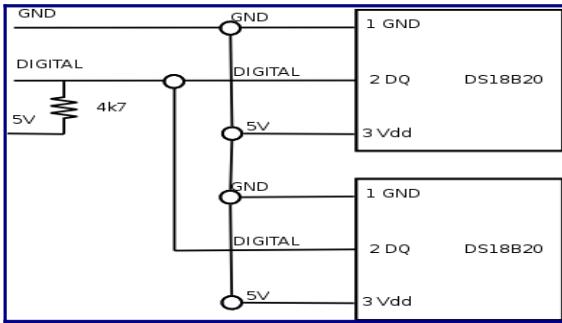


también sin alimentación, con lo que se conoce como alimentación parásita. En este modo de funcionamiento la alimentación la saca directamente de la línea DQ por lo que solo se necesitan dos cables (DQ y GND).

Cada uno de ellos dispone de un identificador único de 64 bits, lo que permite que puedan coexistir varios sensores en un único bus 1-wire. El bus 1-wire puede ser de hasta 100 metros de largo por lo que la colocación de los sensores respecto del módulo no debería ser un problema, además no requiere de elementos externos, el cable de conexión tiene que ser apantallado.



La programación de este dispositivo es muy simple y para ello tenemos una librería específica la “DallasTemperature”.



Una librería es unos archivos con funciones que nos permiten manejar determinadas características del Arduino o de un componente conectado al Arduino.

Estas librerías deben de estar en un subdirectorio dentro del directorio que tiene el IDE de Arduino específicos para ellas llamado “libraries”, si miráis en este directorio veréis que hay muchos subdirectorios de diversas librerías, el IDE del Arduino cada vez que lo ejecutamos mira estos directorios para registrar todas las librerías. Si ejecutamos el IDE del Arduino y nos vamos al menú “Sketch” y a “Import Library” veremos todas las librerías registradas en el IDE, si le damos a una de ellas, nos escribirá en el código la llamada a esta librería para poder usarla.

Conque empecemos ejecutemos el IDE y al arrancar ya tendremos abierto el editor con un nuevo archivo, empezaremos añadiendo dos librerías que nos hace falta para manejar el DS18B20 que son la “OneWire.h” que ya trae el IDE en su instalación y la “DallasTemperature.h” que no descargaremos y copiaremos de este del [Enlace](#).

Para añadirlas lo podemos hacer dándole al menú “Sketch->Import Library-> OneWire” y “Sketch->Import Library DallasTemperature” o simplemente escribiendo

```
#include <OneWire.h>
#include <DallasTemperature.h>
```

A continuación definiremos el pin al que hemos conectado nuestro/s DS18B20 en mi caso al pin 7.

```
#define PIN_P_TEMPER 7
```

Luego declararemos una variable pública que apunte a nuestro controlador/s DS18B20, de esta forma

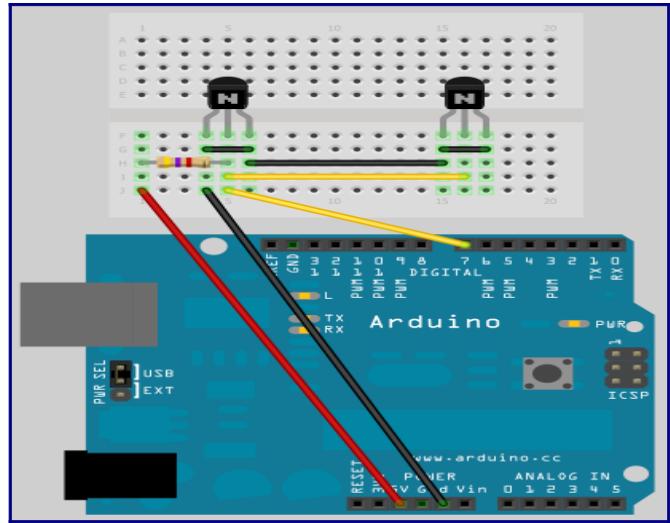
```
// Direcciones de los sensores de temperatura
DeviceAddress AguaTermo, TapaTermo, DepoTermo;
```

Yo aquí declaro 3 variables porque tengo 3 sensores montados, de esta forma cada vez que quiera consultar algunos de mis sensores los llamaré a cada uno por sus nombres.

Luego declaramos las variables que nos permite manejar un objeto de tipo DallasTemperature.

```
OneWire oneWire(PIN_P_TEMPER);
DallasTemperature sensors(&oneWire);
```

De esta forma declaramos una conexión ONEWIRE llamada “oneWire” que apunta a nuestro sensor y esta se la pasamos a “sensors” para crearlo, que es un objeto DallasTemperature, estas dos variables las podéis llamar como vosotros queráis, estos nombres son solo los que yo decidí.



## La función setup()

En ella declararemos que el pin 7 es de entrada de datos

```
void setup()
{
//pin 7 es de entrada de datos
pinMode(PIN_P_TEMPER, INPUT );

//Iniciamos el controlador del controlador/s DS18B20
sensors.begin();

//Compruebo que estén todos los sensores.
//Cuidado, ya que al llamar a "sensors.getAddress()", lo que hacemos es definir que
//variable es cada sensor, por ejemplo definimos que el sensor Nº0 es DepoTermo.
if (sensors.getDeviceCount() < 3) Serial.println ("Faltan sensores de temperatura");
if (!sensors.getAddress(DepoTermo, 0)) Serial.println ("Falta el sensores de temperatura del
Deposito");
if (!sensors.getAddress(AquaTermo, 1)) Serial.println ("Falta el sensores de temperatura del
agua");
if (!sensors.getAddress(TapaTermo, 2)) Serial.println ("Falta el sensores de temperatura de la
tapa");

//definimos sus mínimo y máximo de temperatura para cada uno, de forma que los
//calentadores se enciendan cuando la temperatura baje a los 26.25 grados y se apaguen
//cuando lleguen a los 26.75 grados, y los ventiladores de la tapa se enciendan al llegar
//a los 30.00 grados y se apaguen al llegar a los 29.50 grados.
sensors.setHighAlarmTemp( AquaTermo, 26.75 );
sensors.setLowAlarmTemp( AquaTermo, 26.25 );
sensors.setHighAlarmTemp( DepoTermo, 26.75 );
sensors.setLowAlarmTemp( DepoTermo, 26.25 );
sensors.setHighAlarmTemp( TapaTermo, 30.00 );
sensors.setLowAlarmTemp( TapaTermo, 29.50 );
}
```

Bien con esto ya tenemos definido nuestro inicio del Arduino el cual nos va a iniciar y configurar nuestros sensores.

Ahora veremos como leer la temperatura con nuestros sensores.

```
void loop()
{
float TemperaturaAcuario, TemperaturaDeposito, TemperaturaTapa;

//Llamamos a la función requestTemperatures() de nuestro controlador de sensores
//para que se prepare a nuestra llamada de lectura.
sensors.requestTemperatures();

//llamamos a getTempC() pasándole la variable del sensor que queremos leer
//y esta función nos devolverá la temperatura de el sensor llamado.
TemperaturaAcuario = sensors.getTempC( AquaTermo ); // ACUARIO
TemperaturaTapa = sensors.getTempC( TapaTermo ); // TAPA
TemperaturaDeposito = sensors.getTempC(DepoTermo); // DEPOSITO

//ahora que tenemos las temperaturas de nuestros sensores podemos hacer con
```

```

//ellas los que queramos, por ejemplo sacarlas por una pantalla.

//A continuación comprobaremos si se ha activado alguna alarma de temperatura en nuestros
sensores
if (sensors.hasAlarm(AguaTermo))
{
// Se ha activado una alarma, comprobamos cual y actuamos.
if ( sensors.getTempC(AguaTermo) < sensors.getLowAlarmTemp(AguaTermo) )
{
// Encendemos el calentador del acuario
}
if ( sensors.getTempC(AguaTermo) > sensors.getHighAlarmTemp(AguaTermo) )
{
//Apagamos el calentador del acuario
}
}

if (sensors.hasAlarm(DepoTermo))
{
if ( sensors.getTempC(DepoTermo) < sensors.getLowAlarmTemp(DepoTermo))
{
// Encendemos el calentador del deposito
}
if ( sensors.getTempC(DepoTermo) > sensors.getHighAlarmTemp(DepoTermo))
{
// Apagamos el calentador del deposito
}
}

if (sensors.hasAlarm(TapaTermo))
{
if ( sensors.getTempC(TapaTermo) > sensors.getHighAlarmTemp(TapaTermo))
{
// Encendemos los ventiladores de la tapa
}
if ( sensors.getTempC(TapaTermo) < sensors.getLowAlarmTemp(TapaTermo))
{
// Apagamos los ventiladores de la tapa.
}
}
}

```

En este [enlace](#) tenéis el código.

En la próxima hablaremos de la pantalla LCD importante para ver lo que pasa y optimizaremos este código para que sea más efectivo.

## La pantalla

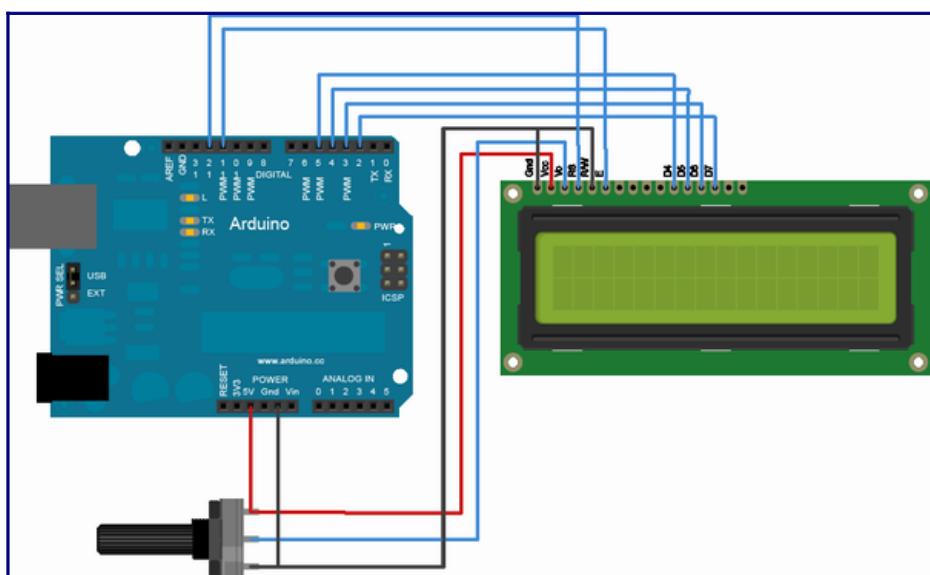
Como ya vimos al principio del post lo normal es que la pantalla sea una LCD de cristal líquido, las más comunes son de caracteres y se definen por cuantos caracteres caben en cada línea y cuantas líneas tiene. Para comunicarnos con ellas tenemos que hacerlo por un bus de datos de 4bit o 8bit en definitiva por 4 cables o 8, los cuales irán conectados a los pines digitales del Arduino, lo normal es que con 4bit se puedan manejar casi todas.

Además del bus de datos tendremos que conectar algunos cables más, como son registro (**RS**) que es el que controla en qué parte de la memoria del LCD estás escribiendo datos, el cual irá a otro pin digital del Arduino, también conectaremos el pin de lectura/escritura (**R/W**) que selecciona el modo de lectura o el de escritura, este a no ser que nos queramos meter en reprogramación de los caracteres de la pantalla lo pondremos a masa (polo negativo).

Luego nos queda el pin para habilitar (**E/enable**) que habilita los registros, el cual irá a otro pin digital del Arduino, también tendremos que conectar un potenciómetro de 10K Ohm conectado entre el positivo y el negativo de la alimentación y con su punto central al pin (**V0**) para regular el contraste de la pantalla.

Y por último los cables de alimentación negativo y positivo de 5v para la pantalla y suele ser también necesario para pines de retroiluminación (**Bklt+** y **Bklt-**), encender y apagar la retro-iluminación.

En definitiva:



El bus de datos

D0 |  
D1 |  
D2 | 8Bit.  
D3 |  
D4 | |  
D5 | 4Bit. |  
D6 | |  
D7 | |

**RS** Control del direccionamiento de memoria.

**RW** Selección de modo lectura/escritura.

**E** Habilitación de los registros.

**V0** Contraste de la pantalla.

**Bklt+** Positivo alimentación de la retro-iluminación.

**Bklt-** Negativo alimentación de la retro-iluminación.

**VDD** Positivo alimentación de la pantalla.

**VSS** Negativo alimentación de la pantalla.

Para conectar esta pantalla aparte de en lo escrito anteriormente tenéis toda la información de conexión y la librería para manejarla en el playground de la página de Arduino, en este enlace <http://arduino.cc/playground/Code/LCD12864>.

La mayoría de estas pantallas están basadas o son compatibles con el controlador **Hitachi HD44780** y para manejarlas el Arduino ya trae una librería llamada LiquidCrystal y que es muy simple de usar.

```
#include <LiquidCrystal.h> //Añadimos la librería en nuestro código.
```

```

//Inicializamos nuestra variable de la pantalla con la librería diciéndole en que pines de control
//hemos conectado la pantalla.
//El orden de los pin es - RS, E, D0, D1, D2, D3
LiquidCrystal MiPantalla(12, 2, 10, 9, 8, 7);

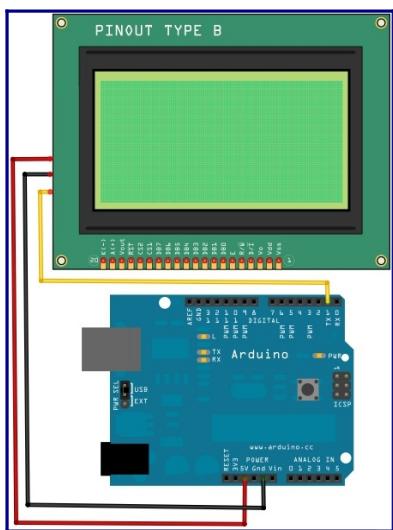
void setup(void)
{
// Ponemos en marcha la pantalla diciéndole cuantos caracteres tiene de largo la pantalla y cuantas
líneas tiene,
//en este caso 16x2 caracteres, el ultimo es siempre por defecto un 1.
MiPantalla.begin(16, 2, 1);
}

void loop(void)
{

//Ponemos el cursor en la posición que queremos escribir, en escala de caracteres y línea.
MiPantalla.setCursor(0,0);
//Pintamos en la posición de la pantalla que nos pusimos el texto.
MiPantalla.print("Hola mundo");
//Nos ponemos ahora en el principio de la segunda línea.
MiPantalla.setCursor(0,0);
//y pintamos el otro mensaje.
MiPantalla.print("Como estas, yo bien.");
}

```

Pero no todas las pantallas tienen tanto “guasa” para conectarla, las hay también que se conectan por serie o lo que es lo mismo con tan solo 4 cables.



Estas pantallas son iguales que las que van conectada en paralelo, como la descrita anteriormente, solo cambia que tienen un modulo conectado a todas sus patillas adosado en su espalda que traduce las ordenes que se le mandan por serie, los 4 cables son positivo y negativo de alimentación y 1 emisor de datos y 1 receptor de datos.



Pantalla con el modulo para comunicación en serie en su espalda.

Para programar esta pantalla nos hará falta otro tipo de librería, la cual siempre dependerá del modulo que tenga de comunicación serie, estas librerías se descargan siempre del mismo sitio donde las compramos y si no con una simple búsqueda de su nombre nos saldrán.

Como ejemplo pondremos la que yo e utilizado en mi montaje una LCD gráfica de 128x64 y serie, esta pantalla tiene su propia librería pero la verdad es que no me gusto nada y me ice la mía propia, mas versátil y efectiva.

Os la podéis descargar de este [enlace](#).

```
#include "LCD128x64.h" //Añadimos la librería en nuestro código.  
//Inicializamos nuestra variable de la pantalla con la librería sin parámetros ningunos.  
LCD128x64Class MiPantalla();
```

Pero tener en cuenta que al funcionar con un puerto serial tenemos que editar la librería y modificarla si deseamos utilizar otro puerto serial que no sea el por defecto del Arduino.  
Por ejemplo si utilizados un Arduino mega y queremos utilizar el puerto serial 1 tendremos que cambiar el “Serial” por “Serial1”.

```
void setup(void)  
{  
  
    // Definimos el tiempo de espera entre cada orden a la pantalla.  
    MiPantalla.SetDelay(200); //Milisegundos  
    // Definimos el brillo que queremos en la pantalla de 0 a 100.  
    MiPantalla.Brillo(70);  
}  
  
void loop(void)  
{  
    //Borramos la pantalla entera.  
    Lcd.Cls();  
  
    //Escribimos en la posición que indiquemos, la posición va en escala de puntos 128 x 64 puntos.  
    MiPantalla.WriteXY( Reloj.DateTime(), 10, 61);  
}
```

Para este tipo de pantallas también tenemos otras órdenes como:

```
//Borrar una zona determinada de la pantalla  
void Borrar(byte desde_x, byte desde_y, byte hasta_x, byte hasta_y);  
  
//Activar o desactivar un punto de la pantalla.  
void Pixel(byte x, byte y, byte modo);  
  
//Posicionar el cursor en unas coordenadas de la pantalla.  
void GotoXY(byte x, byte y);  
  
//Escribir en la pantalla en la posición actual del cursor.  
void Write(char* st);  
void Write(int st);  
void Write(long st);  
void Write(float st);  
void Write(float st, int decimales);  
  
//Escribir en la pantalla en la posición que le digamos.  
void WriteXY(char* st, byte x, byte y);  
void WriteXY(int st, byte x, byte y);  
void WriteXY(long st, byte x, byte y);  
void WriteXY(float st, byte x, byte y);  
void WriteXY(float st, byte x, byte y, int decimales);
```

```

//Dibujar en la pantalla un rectangulo.
void Rectangulo(byte x1, byte y1, byte x2, byte y2);

//Dibujar en la pantalla un circulo.
void Circulo(byte CentroX, byte CentroY, byte Radio, byte modo);

//Dibujar en la pantalla una linea.
void Linea(int x1, int y1, int x2, int y2, int z);

//Dibujar en la pantalla un grafico.
void Imagen( int desde_x, int desde_y, int ancho, int alto, int modo, PROGMEM prog_uint16_t
*array);

```

Bueno pues tras esto nos pondremos a crear ya nuestro montaje inicial, uniendo nuestro sensor de temperatura y nuestra pantalla en un montaje y un código.

Para esto escribiremos el comienzo de nuestro programa, estructurándolo bien para poder seguirlo sin perdernos, ni liarnos.

No puedo dar una clase de programación, pero os aseguro que si me seguís cualquiera de vosotros podréis crear vuestra programación a medida para vuestro montaje.

No es difícil, solo es entretenido y muy gratificante.

Hoy un poco de programación, un par de post antes vimos la programación de los sensores de temperatura, para lo cual puse el siguiente código.

```

#include <OneWire.h>
#include <DallasTemperature.h>

#define PIN_P_TEMPER 7

DeviceAddress AguaTermo, TapaTermo, DepoTermo; // Direcciones de los sensores de temperatura

OneWire oneWire( PIN_P_TEMPER );
DallasTemperature sensors(&oneWire);

void setup()
{
pinMode(PIN_P_TEMPER, INPUT );

//Iniciamos el controlador del controlador/s DS18B20
sensors.begin();

//Compruebo que estén todos los sensores.
//Cuidado, ya que al llamar a "sensors.getAddress()", lo que hacemos es definir que
//variable es cada sensor, por ejemplo definimos que el sensor Nº0 es DepoTermo.
if (sensors.getDeviceCount() < 3) Serial.println( "Faltan sensores de temperatura");
if (!sensors.getAddress(DepoTermo, 0)) Serial.println("Falta el sensores de temperatura del
Deposito");
if (!sensors.getAddress(AguaTermo, 1)) Serial.println("Falta el sensores de temperatura del agua");
if (!sensors.getAddress(TapaTermo, 2)) Serial.println("Falta el sensores de temperatura de la tapa");

//definimos sus mínimo y máximo de temperatura para cada uno, de forma que los
//calentadores se enciendan cuando la temperatura baje a los 26.25 grados y se apaguen

```

```

//cuando lleguen a los 26.75 grados, y los ventiladores de la tapa se enciendan al llegar
// a los 30.00 grados y se apaguen al llegar a los 29.50 grados.
sensors.setHighAlarmTemp( AguaTermo, 26.75 );
sensors.setLowAlarmTemp( AguaTermo, 26.25 );
sensors.setHighAlarmTemp( DepoTermo, 26.75 );
sensors.setLowAlarmTemp( DepoTermo, 26.25 );
sensors.setHighAlarmTemp( TapaTermo, 30.00 );
sensors.setLowAlarmTemp( TapaTermo, 29.50 );
}

void loop()
{
float TemperaturaAcuario, TemperaturaDeposito, TemperaturaTapa;

//Llamamos a la función requestTemperatures() de nuestro controlador de sensores
//para que se prepare a nuestra llamada de lectura.
sensors.requestTemperatures();

//llamamos a getTempC() pasándole la variable del sensor que queremos leer
// y esta función nos devolverá la temperatura de el sensor llamado.
TemperaturaAcuario = sensors.getTempC( AguaTermo ); // ACUARIO
TemperaturaTapa = sensors.getTempC( TapaTermo ); // TAPA
TemperaturaDeposito = sensors.getTempC(DepoTermo); // DEPOSITO

//ahora que tenemos las tempreaturas de nuestros sensores podemos hacer con
//ellas los que queramos, por ejemplo sacarlas por una pantalla.

//A continuación comprobaremos si se a activado alguna alarma de temperatura en nuestros
sensores
if (sensors.hasAlarm(AguaTermo))
{
// Se a activado una alarma, comprobamos cual y actuamos.
if ( sensors.getTempC(AguaTermo) < sensors.getLowAlarmTemp(AguaTermo) )
{
// Encendemos el calentador del acuario
}
if ( sensors.getTempC(AguaTermo) > sensors.getHighAlarmTemp(AguaTermo) )
{
//Apagamos el calentador del acuario
}
}
if (sensors.hasAlarm(DepoTermo))
{
if ( sensors.getTempC(DepoTermo) < sensors.getLowAlarmTemp(DepoTermo))
{
// Encendemos el calentador del deposito
}
if ( sensors.getTempC(DepoTermo) > sensors.getHighAlarmTemp(DepoTermo))
{
// Apagamos el calentador del deposito
}
}
if (sensors.hasAlarm(TapaTermo))
{

```

```

if ( sensors.getTempC(TapaTermo) > sensors.getHighAlarmTemp(TapaTermo))
{
// Encendemos los ventiladores de la tapa
}
if ( sensors.getTempC(TapaTermo) < sensors.getLowAlarmTemp(TapaTermo))
{
// Apagamos los ventiladores de la tapa.
}
}
}
}

```

Este código esta bien pero si seguimos programando así, escribiendo todo de forma seguida dentro de la función “setup()” y “loop()”, cuando llevemos la mitad del código escrito nos será casi imposible seguir el hilo de lo que hace el código, por lo cual lo ideal es repartir el código en funciones, las cuales se declaran igual que “setup()” y “loop()”, pero con otros nombre que nosotros decidamos, hay que tener en cuenta por ejemplo que las variables que declaremos dentro de una función, solo podremos acceder a ella dentro de la función y estas se reiniciarán cada vez que salgamos de la función.

Siguiendo este término, modificaremos el código ya escrito, quitando el código de iniciación de los sensores de temperatura de la función “setup()” y poniendo este código dentro de una función que llamaremos “IniciarTemperaturas()” y dentro de “setup()” pondremos tan solo la llamada a esta función, también el código de comprobación de la temperatura que está en “loop()” lo quitaremos y lo pondremos dentro de otra función que llamaremos “MirarTemperaturas()” y dentro de “loop()” pondremos una llamada a esta función, espero que esto esté claro para los que están pegados en programación. Este nuevo código lo guardaremos con otro nombre como “NuestroControlador.ino”, este fichero será nuestro proyecto de controlador y a partir de ahora le iremos añadiendo todo el código que vallamos avanzando para controlar todo.

```

#include <OneWire.h>
#include <DallasTemperature.h>

#define PIN_P_TEMPER 7

DeviceAddress AguaTermo, TapaTermo, DepoTermo; // Direcciones de los sensores de temperatura

OneWire oneWire( PIN_P_TEMPER );
DallasTemperature sensors(&oneWire);

void setup()
{
pinMode(PIN_P_TEMPER, INPUT );

IniciarTemperaturas(); //Iniciamos los sensores de temperatura.
}

void loop()
{
MirarTemperaturas(); //Miramos la temperaturas
}

void IniciarTemperaturas()
{
//Iniciamos el controlador del controlador/s DS18B20

```

```

sensors.begin();

//Compruebo que estén todos los sensores.
//Cuidado, ya que al llamar a "sensors.getAddress()", lo que hacemos es definir que
//variable es cada sensor, por ejemplo definimos que el sensor N°0 es DepoTermo.
if (sensors.getDeviceCount() < 3) Serial.println( "Faltan sensores de temperatura");
if (!sensors.getAddress(DepoTermo, 0)) Serial.println("Falta el sensores de temperatura del
Deposito");
if (!sensors.getAddress(AguaTermo, 1)) Serial.println("Falta el sensores de temperatura del agua");
if (!sensors.getAddress(TapaTermo, 2)) Serial.println("Falta el sensores de temperatura de la tapa");

//definimos sus mínimo y máximo de temperatura para cada uno, de forma que los
//calentadores se enciendan cuando la temperatura baje a los 26.25 grados y se apaguen
//cuando lleguen a los 26.75 grados, y los ventiladores de la tapa se enciendan al llegar
//a los 30.00 grados y se apaguen al llegar a los 29.50 grados.
sensors.setHighAlarmTemp( AguaTermo, 26.75 );
sensors.setLowAlarmTemp( AguaTermo, 26.25 );
sensors.setHighAlarmTemp( DepoTermo, 26.75 );
sensors.setLowAlarmTemp( DepoTermo, 26.25 );
sensors.setHighAlarmTemp( TapaTermo, 30.00 );
sensors.setLowAlarmTemp( TapaTermo, 29.50 );
}

void MirarTemperaturas()
{
float TemperaturaAcuario, TemperaturaDeposito, TemperaturaTapa;

//Llamamos a la función requestTemperatures() de nuestro controlador de sensores
//para que se prepare a nuestra llamada de lectura.
sensors.requestTemperatures();

//llamamos a getTempC() pasándole la variable del sensor que queremos leer
//y esta función nos devolverá la temperatura de el sensor llamado.
TemperaturaAcuario = sensors.getTempC( AguaTermo ); // ACUARIO
TemperaturaTapa = sensors.getTempC( TapaTermo ); // TAPA
TemperaturaDeposito = sensors.getTempC(DepoTermo); // DEPOSITO

//ahora que tenemos las tempreaturas de nuestros sensores podemos hacer con
//ellas los que queramos, por ejemplo sacarlas por una pantalla.

//A continuación comprobaremos si se a activado alguna alarma de temperatura en nuestros
sensores
if (sensors.hasAlarm(AguaTermo))
{
// Se a activado una alarma, comprobamos cual y actuamos.
if ( sensors.getTempC(AguaTermo) < sensors.getLowAlarmTemp(AguaTermo) )
{
// Encendemos el calentador del acuario
}
if ( sensors.getTempC(AguaTermo) > sensors.getHighAlarmTemp(AguaTermo) )
{
//Apagamos el calentador del acuario
}
}

```

```

if (sensors.hasAlarm(DepoTermo))
{
if ( sensors.getTempC(DepoTermo) < sensors.getLowAlarmTemp(DepoTermo))
{
// Encendemos el calentador del deposito
}
if ( sensors.getTempC(DepoTermo) > sensors.getHighAlarmTemp(DepoTermo))
{
// Apagamos el calentador del deposito
}
}
if (sensors.hasAlarm(TapaTermo))
{
if ( sensors.getTempC(TapaTermo) > sensors.getHighAlarmTemp(TapaTermo))
{
// Encendemos los ventiladores de la tapa
}
if ( sensors.getTempC(TapaTermo) < sensors.getLowAlarmTemp(TapaTermo))
{
// Apagamos los ventiladores de la tapa.
}
}
}
}

```

Aquí tenéis el código final [NuestroControlador001.zip](#)

Ahora le añadiremos el código para empezar a pintar en nuestra pantalla, para lo cual añadiremos a nuestro código la librería de nuestra pantalla y su iniciación.

Por ejemplo para una pantalla de caracteres.

```

#include <OneWire.h>
#include <DallasTemperature.h>
#include <LiquidCrystal.h>

#define PIN_P_TEMPER 7
#define LCD_WIDTH 16 //Ancho de la pantalla
#define LCD_HEIGHT 2 // Alto de la pantalla

DeviceAddress AguaTermo, TapaTermo, DepoTermo; // Direcciones de los sensores de temperatura

OneWire oneWire( PIN_P_TEMPER );
DallasTemperature sensors(&oneWire);
LiquidCrystal lcd(12, 2, 10, 9, 8, 7); //Iniciamos la pantalla

void setup()
{
pinMode(PIN_P_TEMPER, INPUT );

lcd.begin(LCD_WIDTH, LCD_HEIGHT,1); //Arrancamos nuestra pantalla

IniciarTemperaturas(); //Iniciamos los sensores de temperatura.
}

```

Y en la función “MirarTemperaturas()” pondremos el código para que pinte los valores de las temperaturas.

```
void MirarTemperaturas()
{
float TemperaturaAcuario, TemperaturaDeposito, TemperaturaTapa;
char data[12]; //Variable para pasar el valor de la temperatura a texto.

//Llamamos a la función requestTemperatures() de nuestro controlador de sensores
//para que se prepare a nuestra llamada de lectura.
sensors.requestTemperatures();

//llamamos a getTempC() pasándole la variable del sensor que queremos leer
//y esta función nos devolverá la temperatura del sensor llamado.
TemperaturaAcuario = sensors.getTempC(AguaTermo); // ACUARIO
TemperaturaTapa = sensors.getTempC(TapaTermo); // TAPA
TemperaturaDeposito = sensors.getTempC(DepoTermo); // DEPOSITO

//ahora que tenemos las temperaturas de nuestros sensores las pintamos en nuestra pantalla
lcd.setCursor(0,0); //Nos ponemos en el inicio de la pantalla
lcd.print("AGUA: ");
dtostrf(data, 4, 2, TemperaturaAcuario); //Convertimos el valor de la temperatura a texto con dos decimales.
lcd.print(data); // Lo pintamos.
lcd.setCursor(11,0); //Nos ponemos en la derecha de la pantalla
lcd.print("TAPA: ");
dtostrf(data, 4, 2, TemperaturaTapa); //Convertimos el valor de la temperatura a texto con dos decimales.
lcd.print(data); // Lo pintamos.
lcd.setCursor(0,1); //Nos ponemos al inicio de la segunda línea.
lcd.print("DEPO: ");
dtostrf(data, 4, 2, TemperaturaDeposito); //Convertimos el valor de la temperatura a texto con dos decimales.
lcd.print(data); // Lo pintamos.
```

Con la pantalla 128x64 conectada en paralelo.

```
#include <OneWire.h>
#include <DallasTemperature.h>
#include "LCD12864R.h"
#define AR_SIZE( a ) sizeof( a ) / sizeof( a[0] )

#define PIN_P_TEMPER 7

DeviceAddress AguaTermo, TapaTermo, DepoTermo; // Direcciones de los sensores de temperatura

OneWire oneWire( PIN_P_TEMPER );
DallasTemperature sensors(&oneWire);

void setup()
{
pinMode(PIN_P_TEMPER, INPUT );

LCDA.Initialise(); //Iniciamos la pantalla
```

```

LCDA.Clear(); //Limpiamos la pantalla

IniciarTemperaturas(); //Iniciamos los sensores de temperatura.
}

```

Y ahora lo pintamos

```

//ahora que tenemos las temperaturas de nuestros sensores la pintamos en nuestra pantalla
LCDA.DisplayString(0,1, "AGUA: ", AR_SIZE("AGUA: "));
dtostrf( data, 4, 2, TemperaturaAcuario ); //Convertimos el valor de la temperatura a texto con dos decimales.
LCDA.DisplayString(0,8, data, AR_SIZE(data)); // Lo pintamos.
delay(100);
LCDA.DisplayString(1,1, "TAPA: ", AR_SIZE("TAPA: "));
dtostrf( data, 4, 2, TemperaturaTapa ); //Convertimos el valor de la temperatura a texto con dos decimales.
LCDA.DisplayString(1,8, data, AR_SIZE(data)); // Lo pintamos.
delay(100);
LCDA.DisplayString(1,1, "DEPO: ", AR_SIZE("DEPO: "));
dtostrf( data, 4, 2, TemperaturaDeposito ); //Convertimos el valor de la temperatura a texto con dos decimales.
LCDA.DisplayString(1,8, data, AR_SIZE(data)); // Lo pintamos.
delay(100);

```

Con la pantalla 128x64 conectada por serial.

```

#include <OneWire.h>
#include <DallasTemperature.h>
#include "LCD128x64.h" // Libreria del LCD

#define PIN_P_TEMPER 7

DeviceAddress AguaTermo, TapaTermo, DepoTermo; // Direcciones de los sensores de temperatura

OneWire oneWire( PIN_P_TEMPER );
DallasTemperature sensors(&oneWire);
LCD128x64Class Lcd( 19, 18 ); //Declaramos la pantalla

void setup()
{
pinMode(PIN_P_TEMPER, INPUT );

Lcd.SetDelay(200); //Declaramos el tiempo en de espera entre cada orden.
Lcd.Brillo(70); //Definimos el brillo que queremos en nuestra pantalla.

IniciarTemperaturas(); //Iniciamos los sensores de temperatura.
}

```

Y ahora lo pintamos

```

//ahora que tenemos las temperaturas de nuestros sensores la pintamos en nuestra pantalla
Lcd.WriteXY("ACUARIO :", 2, 50);
Lcd.WriteXY(TemperaturaAcuario, 57, 50 );

```

```
Lcd.WriteLine("TAPADERA:", 2, 40);
Lcd.WriteLine(TemperaturaTapa, 57, 40 );
Lcd.WriteLine("DEPOSITO:", 2, 30);
Lcd.WriteLine(TemperaturaDeposito, 57, 30 );
```

Aquí tenéis el código final [NuestroControlador002.zip](#)

Dentro de poco pondré la parte del reloj y el teclado, pero mientras tanto para aclarar un poco las ideas a los que están pensando como montar su propio controlador, le pongo unos enlaces de otra mucha gente que esta montando también sus controladores, esperando que podáis coger ideas de como conectar y encajar el controlador en vuestro acuario.

En español:

<http://www.carballada.com/wordpress...-de-acuario-03b>  
<http://code.google.com/p/acuarino/>  
<http://arduino.trucados.com/?cat=1>  
<http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1264144531>  
<http://www.todomarino.com/forum/for...ller-de-Arduino>

En inglés:

[http://reefprojects.com/wiki/Main\\_Page](http://reefprojects.com/wiki/Main_Page)  
<http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1240000018>  
<http://www.nano-reef.com/forums/index.php?showtopic=196161>  
<http://www.nzmas.co.nz/forum/viewtopic.php?f=7&t=10339>  
<http://www.marineaquariumsa.com/showthread.php?t=26372>  
<http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1291521243>  
<http://www.thereefuge.com/threads/a...nd-answers.745/>

Veréis que la gente monta su controlador desde en un taperware hasta en una caja echa en madera.

## Reloj de tiempo real

Hoy veremos como conectar y programar el modulo del reloj para tener controlado la fecha y la hora, como ya vimos anteriormente este es un modulo muy pequeño el cual tiene 5 conexiones de las cuales solamente usaremos 4 que son el positivo, el negativo, SCL y SDA, estas dos ultimas son las correspondiente a una conexión del protocolo I2C, las cuales para un Arduino Uno son los pines analógicos SDA = 4 y SCL = 5 en cambio en el Arduino Mega los pines don el SDA = 20 y SCL = 21.

Para manejar este reloj hay varias librerías en Internet pero la verdad es que después de probar varias (sin gustarme ninguna) me decidí por crear me la mía propia un poco mas cómoda de usar y mas funcional.

Os la podéis descargar de este enlace [RelojDS1307](#).

Una descripción rápida de sus funciones :

```
void SetDateTime( int monthDay, int month, int year, int hour, int minute, int second ); //Establece  
la la fecha y hora del reloj  
void GetDateTime( int &monthDay, int &month, int &year, int &hour, int &minute, int &second );  
//Obtiene la fecha y hora en parámetros separados.  
char * DateTIme();//Obtiene una cadena de texto con la fecha y la hora.  
char * Date();//Obtiene una cadena de texto con la fecha.  
char * Time();//Obtiene una cadena de texto con la hora y minutos.
```

```

int GetHour(); //Obtiene un entero con el valor de la hora.
int GetMinute(); //Obtiene un entero con el valor de los minutos.
int GetYear(); //Obtiene un entero con el valor del año.
int GetMonth(); //Obtiene un entero con el valor del mes.
int GetDay(); //Obtiene un entero con el valor del día.
void SetDiaSemana( int DiaSemana ); //Establece el día de la semana(Lunes,Martes....).
int GetDiaSemana(); //Obtiene un entero con el valor del día de la semana
char * DiaName(); //Obtiene una cadena con el nombre del día de la semana.
char * MesName(); //Obtiene una cadena con el nombre del mes.
boolean IsLater( int hour, int minute ); //No devuelve si la hora actual es mas tarde que la pasada.
boolean IsPrevious( int hour, int minute ); //No devuelve si la hora actual es mas temprana que la pasada.

```

Su uso viendo las funciones que tiene es bastante simple, empezaremos añadiendo en nuestro último código la llamada a la librería <Wire.h> que también utiliza internamente.

```

#include <Wire.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include "LCD128x64.h" // Libreria del LCD
#include "RelojDS1307.h" // Libreria del Reloj

```

Así mismo declararemos la variable con la que utilizaremos el RelojDS1307, en este caso simplemente la llamaremos “Reloj”.

```

OneWire oneWire( PIN_P_TEMPER );
DallasTemperature sensors(&oneWire);
LCD128x64Class Lcd( 19, 18 ); //Declaramos la pantalla
RelojDS1307Class Reloj; //Declaración de la variable del Reloj

```

En la función “setup()” añadiremos la iniciación del sistema wire para uso de nuestro Reloj

```

Lcd.SetDelay(200); //Declaramos el tiempo en de espera entre cada orden.
Lcd.Brillo(70); //Definimos el brillo que queremos en nuestra pantalla.
Wire.begin();

```

También añadiremos otra línea para poner en hora nuestro reloj, pero esta línea será solo para que se use una sola vez, de forma que cuando se reinicie el Arduino y funcione le cargaremos otra vez el mismo código pero sin esta línea, ya que esta línea te pondrá la fecha y hora del reloj en cuanto se inicie el Arduino y si lo apagas y inicias otra vez te la volverá a configurar la misma y entonces tendrá retraso el reloj como es normal.

```

Lcd.SetDelay(200); //Declaramos el tiempo en de espera entre cada orden.
Lcd.Brillo(70); //Definimos el brillo que queremos en nuestra pantalla.
Wire.begin();
Reloj.SetDateTime( 18, 01, 12, 18, 12, 00 );

```

Mas adelante veremos como configurar la hora sin tener que armar este jaleo.

Luego para que podamos ver en nuestra pantalla la fecha y la hora actual, es tan simple como poner en nuestra función “loop()” una llamada a la función de pintar de nuestra pantalla pesándole lo que nos devuelve la función Reloj.DateTime().

```
void loop()
```

```

{
  Lcd.WriteXY( Reloj.DateTime(), 10, 61);
  MirarTemperaturas(); //Miramos la temperaturas
}

```

Esto nos imprimirá en nuestra pantalla algo como “18/01/12 16:35:25”

Aquí os dejo el código de nuestro controlador modificado [NuestroControlador003.zip](#).

En la próxima veremos como guardar y leer nuestra configuración desde la Eprom del Arduino, en esta memoria, aunque le cortemos la corriente al Arduino, no se perderá lo que guardemos.

## Guardar y leer nuestra configuración desde la Eprom del Arduino

Vamos a ver ahora como guardar y leer los datos de nuestra configuración, como vimos anteriormente con nuestro sensor de temperatura podemos controlar los calentadores, pero el valor de temperatura que deseamos tener si lo ponemos en el código escrito, significa que cada vez que queramos cambiar este valor deberemos editar el programa y cargarlo en nuestro Arduino.

Esto no es practico, por lo que lo razonable es que el valor de temperatura deseado este en una variable la cual si queremos cambiar, lo podemos hacer desde un menú en la pantalla, el problema que tiene esto es que si se corta la luz a nuestro Arduino se le borra el valor de todas las variable como es natural, para solucionar este problema tenemos la solución muy practica, que es guardar los datos de nuestra configuración en la EPROM la memoria no volátil del Arduino que tiene 512 bytes de memoria, es poco pero para nuestra configuración nos sobra.

Para poder escribir y leer en la EPROM del Arduino tenemos la librería EEPROM.h la cual ya viene con el IDE del Arduino, esta librería es muy simple ya que solo tiene 2 funciones read() y write() , con lo cual queda claro su manejo.

Pero escribir en una EPROM tiene un problema, que no es como un disco de ordenador, no hay archivos ni directorios, solo un espacio de 512 bytes, por lo que si queremos guardar varios datos tenemos que saber la posición de cada uno, para evitar esto utilizaremos un tipo de dato llamado struct (estructura) que es un grupo de variables unidas por una estructura de esta forma.

```

// Estructura de los datos de configuración
typedef struct Configuracio_struct
{
  float TermoAcuario; //Temperatura Acuario
  float PH; //Valor del PH
  int LuzH_ON; //Hora de encendido de la luz
  int LuzM_ON; //Minutos de encendido de la luz
} Stru_Configuracion;

```

De esta forma solo tendremos que guardar o leer en la EPROM la variable que definamos como Stru\_Configuracion, conteniendo esta todas las variable de nuestra configuración.  
Implementémoslo en nuestro código, añadiendo primero la librería a nuestro código.

```

#include <Wire.h>
#include <OneWire.h>
#include <DallasTemperature.h> // Librería de los sensores de temperatura

```

```
#include <EEPROM.h> // Librería para poder escribir en la eprom del Arduino
#include "RelojDS1307.h" // Librería del Reloj
#include "LCD128x64.h" // Librería del LCD
```

Ahora declaremos el tipo de estructura que tendrá nuestra configuración.

```
// Estructura de los datos de configuración
typedef struct Configuracio_struct
{
    float TermoAcuario; //Temperatura Acuario
    float TermoTapa; //Temperatura Tapa
    float TermoDeposito; //Temperatura Deposito
    float PH; //Valor del PH
    int Luz1H_ON; //Hora de encendido de la luz N°1
    int Luz1M_ON; //Minutos de encendido de la luz N°1
    int Luz1H_OFF; //Hora de apagado de la luz N°1
    int Luz1M_OFF; //Minutos de apagado de la luz N°1
    int Luz2H_ON; //Hora de encendido de la luz N°2
    int Luz2M_ON; //Minutos de encendido de la luz N°2
    int Luz2H_OFF; //Hora de apagado de la luz N°2
    int Luz2M_OFF; //Minutos de apagado de la luz N°2
    int Luz3H_ON; //Hora de encendido de la luz N°3
    int Luz3M_ON; //Minutos de encendido de la luz N°3
    int Luz3H_OFF; //Hora de apagado de la luz N°3
    int Luz3M_OFF; //Minutos de apagado de la luz N°3
} Stru_Configuracion;
```

Y luego declaramos la variable de tipo Stru\_Configuracion.

```
Stru_Configuracion CFG;
```

Ahora para poder asignar o leer un valor de nuestra configuración tan solo tendremos que hacer esto.

```
CFG.TermoAcuario = 27.00;
CFG.PH = 6.8;
```

Para que en caso de que no podamos leer la EPROM o sea la primera vez que lo hagamos, le daremos a nuestra estructura de configuración unos valores por defecto.

```
CFG.TermoAcuario = 27;
CFG.TermoTapa = 30.0;
CFG.TermoDeposito = 20.0;
CFG.PH = 7.0;
CFG.Luz1H_ON = 15;
CFG.Luz1M_ON = 0;
CFG.Luz1H_OFF = 23;
CFG.Luz1M_OFF = 30;
CFG.Luz2H_ON = 15;
CFG.Luz2M_ON = 30;
CFG.Luz2H_OFF = 24;
CFG.Luz2M_OFF = 0;
CFG.Luz3H_ON = 23;
CFG.Luz3M_ON = 50;
```

```
CFG.Luz3H_OFF = 26;
CFG.Luz3M_OFF = 0;
```

Luego escribiremos las funciones que leerá y grabara nuestra configuración en la EEPROM

```
///////////
// GRAVA LA CONFIGURACION EN LA EPROM
//
///////////
void SalvarConfig()
{
    uint8_t Valor[sizeof(CFG)];

    memcpy( &Valor, &CFG, sizeof(CFG));

    for (int i = 0; i <= sizeof(Valor); i++)
        EEPROM.write(i, Valor[i]);
}

///////////
// LEE LA CONFIGURACION DE LA EPROM
//
///////////
void LeerConfig()
{
    uint8_t Valor[sizeof(CFG)];

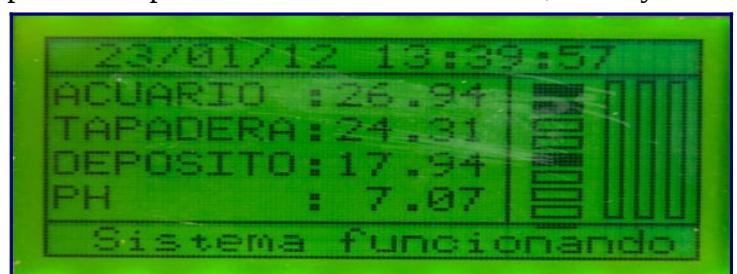
    for (int i = 0; i <= sizeof(CFG); i++)
        Valor[i] = EEPROM.read(i);

    memcpy( &CFG, &Valor, sizeof(CFG));
}
```

Luego todo es tan simple como que al iniciar el programa llamemos a la función LeerConfig() con lo cual tendremos en la variable CFG la estructura con los datos de nuestra configuración, luego si cambiamos algún valor de nuestra configuración llamaremos a SalvarConfig() y tendremos los cambios guardados.

Aquí os dejo el código de nuestro controlador modificado. [NuestroControlador004.zip](#)

Bueno vamos a ver como optimizar el código de pintado en pantalla de nuestro controlador, como ya dijimos las pantallas LCD no se borran solas, por lo que si pintamos algo en ellas no desaparece hasta que pintemos encima o lo borremos, por lo cual tenemos que saber siempre que pintamos y en donde pintamos, para esto lo mejor es hacernos un esquema o dibujo de como queremos que quede nuestra pantalla, teniendo en cuenta su tamaño y limitaciones.

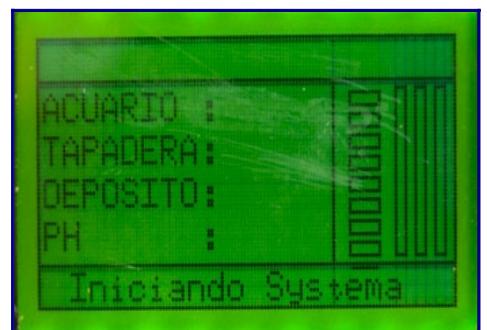


Esta es mi pantalla tal como la diseñe y esta ahora, si nos fijamos bien en ella veremos que esta dividida en varias partes.



Si os fijáis hay ciertas parte que no están marcado, estas son las parte estáticas o lo que es lo mismo las que no cambian, lo mas efectivo es que todas esas partes se pinten desde una función ya que mientras no cambiemos de pantalla, como verán a continuación

Al fotografiarla se me escapo el mensaje del pie, pero quitando esto podéis ver lo que pinta mi función de pintado de pantalla principal, lo demás se pinta desde cada función encargada del valor, por ejemplo la temperatura se pinta en la función que llamamos para leerla `MirarTemperaturas()` sabiendo siempre el lugar donde pintarlo y el espacio que tenemos para ello, esto es importante los valores variables que pintamos tenemos que preocuparnos que siempre tengan el mismo tamaño, para que machaque lo que había antes.



Vamos a aplicar esto en nuestro código, lo primero crearemos 2 nuevas funciones la que pintara las partes estáticas de nuestra pantalla y otra de apoyo que se encargara de pintar avisos y estado en el pie de la pantalla.

```
///////////
// PINTA UN MENSAJE EN EL PIE DE LA PANTALLA
//
/////////
void MsgPie( char * msg )
{
    Lcd.WriteXY( msg, 10, 8);
}

///////////
// PINTA LA PANTALLA PRINCIPAL
//
/////////
void PrintPantMain()
{
    Lcd.Cls();
    Lcd.Rectangulo(0, 0, 127, 63);
    Lcd.Linea( 0, 53, 127, 53, 1);
    Lcd.Linea( 90, 10, 90, 53, 1);
    Lcd.Linea( 0, 10, 127, 10, 1);
    Lcd.WriteXY("ACUARIO :", 2, 50); // Ancho, Alto, Modo
```

```

Lcd.WriteXY("TAPADERA:", 2, 40);
Lcd.WriteXY("DEPOSITO:", 2, 30);
Lcd.WriteXY("PH :", 2, 20);
}

```

Y pondremos la llamada a la función de pintar pantalla al inicio del programa.

```

void setup()
{
pinMode(PIN_P_TEMPER, INPUT);

Lcd.SetDelay(200); //Declaramos el tiempo en de espera entre cada orden.
Lcd.Brillo(70); //Definimos el brillo que queremos en nuestra pantalla.

PrintPantMain();
MsgPie("Iniciando Sistema");

Wire.begin();
.....
.....
.....
IniciarTemperaturas(); //Iniciamos los sensores de temperatura.
MsgPie("Sistema funcionando");
}

```

Ya solo nos queda quitar el texto estático que estábamos pintando en la pantalla y que esta repartido por el código.

```

void MirarTemperaturas()
{
float TemperaturaAcuario, TemperaturaDeposito, TemperaturaTapa;

//Llamamos a la función requestTemperatures() de nuestro controlador de sensores
//para que se prepare a nuestra llamada de lectura.
sensors.requestTemperatures();

//llamamos a getTempC() pasándole la variable del sensor que queremos leer
//y esta función nos devolverá la temperatura de el sensor llamado.
TemperaturaAcuario = sensors.getTempC(AguaTermo); // ACUARIO
TemperaturaTapa = sensors.getTempC(TapaTermo); // TAPA
TemperaturaDeposito = sensors.getTempC(DepoTermo); // DEPOSITO

//ahora que tenemos las temperaturas de nuestros sensores la pintamos en nuestra pantalla
Lcd.WriteXY("ACUARIO:", 2, 50);
Lcd.WriteXY(TemperaturaAcuario, 57, 50 );
Lcd.WriteXY("TAPADERA:", 2, 40);
Lcd.WriteXY(TemperaturaTapa, 57, 40 );
Lcd.WriteXY("DEPOSITO:", 2, 30);
Lcd.WriteXY(TemperaturaDeposito, 57, 30 );

```

Quitaremos el código que esta en azul.

Aquí os dejo el código de nuestro controlador modificado. [NuestroControlador004.zip](#)

## Los relés

Los relés como ya os puse en un [post anterior](#) son un componente esencial en nuestro controlador.

Para poder controlarlos usaremos los pines digitales del Arduino de la siguiente forma

```
digitalWrite( PIN_CONECTADO_EL_RELÉ, HIGH );
```

con esto ponemos en **HIGH** (activo positivo, 5v) el pin donde esta conectado el relé y si le ponemos

```
digitalWrite( PIN_CONECTADO_EL_RELÉ, LOW );
```

lo ponemos en **LOW** (activo negativo, 0v) con lo cual lo apagamos.

Pero hay una pega y es que si utilizamos las placas de relés chinas anteriormente descritas, esta son de [activación negativa](#), o lo que es lo mismo que los relés se activan con **LOW** y se desactivan con **HIGH**, los chinos son raros hasta para esto.

Como esto puede ser un poco lioso a la hora de programar, por lo cual a la hora de hacer mi programa me decidí por programar una función que gestionara la activación negativa de los relés y de paso pintar en la pantalla unos cuadritos representando que relés están activos y cuales no.

```
///////////
// ACTIVA O DESACTIVA UN RELE Y LO PINTA EN LA PANTALLA
// 
void SetRele( int Index, int Pin, boolean Estado )
{
    int I = 8-Index;

    digitalWrite( Pin, !Estado );

    if (Estado)
    {
        Lcd.Rectangulo( 96, 49-(I*5), 103, 48-(I*5));
    }
    else
    {
        Lcd.Linea( 96, 49-(I*5), 103, 49-(I*5), 0);
        Lcd.Linea( 96, 48-(I*5), 103, 48-(I*5), 0);
    }
}

///////////
// PINTAR ESTADO DE LOS RELE EN LA PANTALLA PRINCIPAL AL REPINTARLA
// 
void RePintarRelé()
{
    byte Pines[] = {42, 40, 38, 36, 34, 30, 32, 28};
    int I;
```

```

for( I=8; I>1; I-- )
{
if (!digitalRead(Pines[I]))
{
Lcd.Rectangulo( 96, 49-(I*5), 103, 48-(I*5));
}
else
{
Lcd.Linea( 96, 49-(I*5), 103, 49-(I*5), 0);
Lcd.Linea( 96, 48-(I*5), 103, 48-(I*5), 0);
}
}
}
}

```

El manejo de estas funciones es simple, el primer parámetro es el índice indicador del relé, en mi caso como tengo 8 y los pinto verticalmente el primero de abajo es el 1 y el ultimo de arriba el 8, el segundo parámetro es el pin donde esta conectado y el tercero el estado que le queremos dar, la función ya se preocupa de la [activación negativa](#) de los relés chinos.

```

SetRele( 1, PIN_CONECTADO_EL_RELE, HIGH ); //Activamos y pintamos el relé N°1

SetRele( 1, PIN_CONECTADO_EL_RELE, LOW ); //Desactivamos y pintamos el relé N°1

```

Luego teniendo en cuenta que tendremos que poner mas adelante otras pantallas para configurar el controlador, tenemos otra función que pintara en la pantalla el estado de todos los relés, cuando volvamos a pintar la pantalla principal.

```
RePintarRele();
```

Ya solo nos queda implementarlo en el código de nuestro controlador, lo primero definir los pines donde tenemos conectado nuestros relés.

```

#define PIN_D_RELÉ_CALENTADOR 28 //8
#define PIN_D_RELÉ_CALENTDEPO 32 //7
#define PIN_D_RELÉ_EXTRATORES 30 //6
#define PIN_D_RELÉ_LUCES 34 //5
#define PIN_D_RELÉ_CO2 36 //4
#define PIN_D_RELÉ_OSMOSIS 38 //3
#define PIN_D_RELÉ_LLUVIA 40 //2
#define PIN_D_RELÉ_LLLENADO 42 //1

```

Luego en el setup() configurar estos pines como de salida.

```

pinMode( PIN_D_RELÉ_CALENTADOR, OUTPUT );
pinMode( PIN_D_RELÉ_CALENTDEPO, OUTPUT );
pinMode( PIN_D_RELÉ_EXTRATORES, OUTPUT );
pinMode( PIN_D_RELÉ_LUCES, OUTPUT );
pinMode( PIN_D_RELÉ_CO2, OUTPUT );
pinMode( PIN_D_RELÉ_OSMOSIS, OUTPUT );
pinMode( PIN_D_RELÉ_LLUVIA, OUTPUT );
pinMode( PIN_D_RELÉ_LLLENADO, OUTPUT );

```

Pero aquí con los relés chinos al ser de activación negativa, lo que pasara es que al definir los pines como de salida el Arduino los pondrá a negativo ( 0v ) con lo cual activara todos los relés para corregir esto pondremos un bucle FOR para desactivarlos, esto será muy rápido por lo que no le dará tiempo a funcionar ningún aparato conectado.

```
byte n;
for(n=28; n < 43; n = n + 2)
{
  digitalWrite( n, HIGH);
}
```

La función RePintarRele() la tenemos que poner en nuestra función PrintPantMain() para que pinte los recuadros de nuestros relés al pintar la pantalla.

```
void PrintPantMain()
{
Lcd.Cls();
Lcd.Rectangulo(0, 0, 127, 63);
Lcd.Linea( 0, 53, 127, 53, 1);
Lcd.Linea( 90, 10, 90, 53, 1);
Lcd.Linea( 0, 10, 127, 10, 1);
Lcd.WriteXY("ACUARIO:", 2, 50); // Ancho, Alto, Modo
Lcd.WriteXY("TAPADERA:", 2, 40);
Lcd.WriteXY("DEPOSITO:", 2, 30);
Lcd.WriteXY("PH :", 2, 20);
RePintarRele();
}
```

Bien con esto ya tenemos definido todo lo necesario para poner en funcionamientos nuestros relés, empezaremos por los relés que controlan los calentadores, en la función PrintTemperatura().

```
if (sensors.hasAlarm(AguaTermo))
{
if ( sensors.getTempC(AguaTermo) < sensors.getLowAlarmTemp(AguaTermo) &&
digitalRead(PIN_D_RELLE_CALENTADOR))
SetRele( 8, PIN_D_RELLE_CALENTADOR, HIGH );
if ( sensors.getTempC(AguaTermo) > sensors.getHighAlarmTemp(AguaTermo) && (!
digitalRead(PIN_D_RELLE_CALENTADOR)))
SetRele( 8, PIN_D_RELLE_CALENTADOR, LOW );
}
if (sensors.hasAlarm(DepoTermo))
{
if ( sensors.getTempC(DepoTermo) < sensors.getLowAlarmTemp(DepoTermo) &&
digitalRead(PIN_D_RELLE_CALENTDEPO))
SetRele( 7, PIN_D_RELLE_CALENTDEPO, HIGH );
if ( sensors.getTempC(DepoTermo) > sensors.getHighAlarmTemp(DepoTermo) && (!
digitalRead(PIN_D_RELLE_CALENTDEPO)))
SetRele( 7, PIN_D_RELLE_CALENTDEPO, LOW );
}
if (sensors.hasAlarm(TapaTermo))
{
if ( sensors.getTempC(TapaTermo) > sensors.getHighAlarmTemp(TapaTermo) &&
digitalRead(PIN_D_RELLE_EXTRATORES))
```

```

SetRele( 6, PIN_D_RELÉ_EXTRATORES, HIGH );
if ( sensors.getTempC(TapaTermo) < sensors.getLowAlarmTemp(TapaTermo) && (!
digitalRead(PIN_D_RELÉ_EXTRATORES)))
SetRele( 6, PIN_D_RELÉ_EXTRATORES, LOW );
}

```

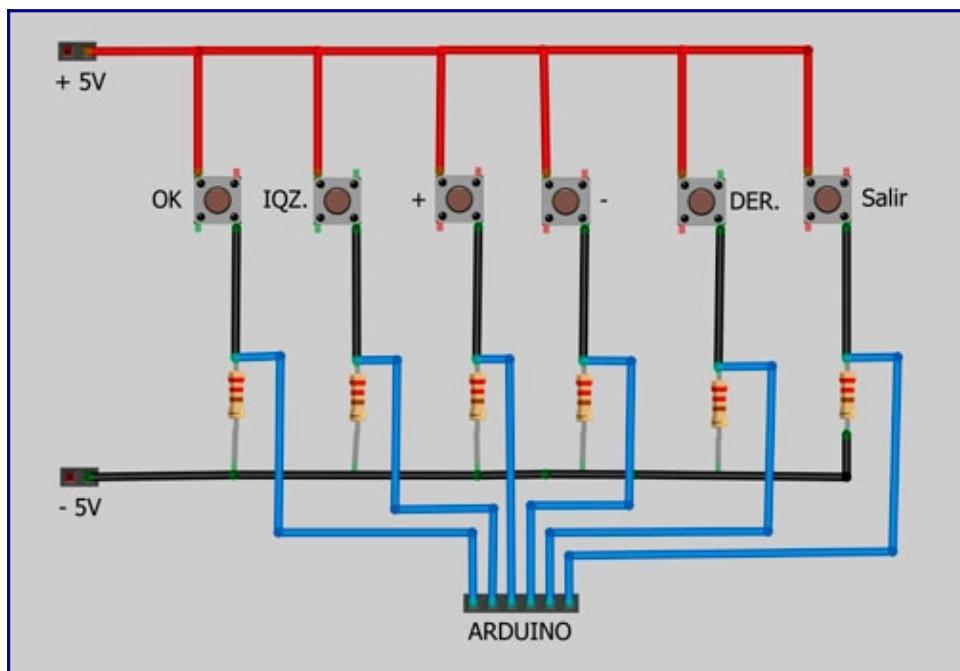
Como veréis utilizo la función digitalRead(PIN) para averiguar si esta o no activo ya el relé.

El código modificado del controlador lo tenéis aquí. [NuestroControlador006.zip](#)

## El teclado.

El teclado es un componente importantísimo para poder interactuar con el Arduino y como vimos en este post hay varias formas de componerlo.

Yo me decidí por este, 2 botones para moverme por la pantalla, 2 botones para aumentar o disminuir los valores, 1 para el OK y 1 para salir de los menús.



Esto lo que hace es poner activo (5v) el pin que este conectado el botón pulsado y lo mismo que usábamos para saber si estaba activo un relé, lo usaremos para saber si esta activo un pulsador, que era la función digitalRead(PIN).

Empecemos a ir definiendo el código de nuestro teclado, empecemos definiendo los pines de nuestro teclado.

```

#define PIN_D_KEY_OK 39
#define PIN_D_KEY_DER 41
#define PIN_D_KEY_SALIR 43
#define PIN_D_KEY_IQZ 45
#define PIN_D_KEY_MAS 47
#define PIN_D_KEY_MENOS 49

```

Ahora en el setup() definámoslo como pines de entrada.

```
pinMode( PIN_D_KEY_OK, INPUT );
pinMode( PIN_D_KEY_DER, INPUT );
pinMode( PIN_D_KEY_SALIR, INPUT );
pinMode( PIN_D_KEY_IZQ, INPUT );
pinMode( PIN_D_KEY_MAS, INPUT );
pinMode( PIN_D_KEY_MENOS, INPUT );
```

Para empezar a usar nuestro teclado crearemos un menú con varias opciones de configuración, para lo cual primero crearemos una función que pinte este menú.

```
///////////
//
// PINTA PANTALLA DE CONFIGURACION
//
/////////
void PrintPantConfig()
{
Lcd.Cls();
Lcd.Rectangulo(0, 0, 127, 63);
Lcd.WriteXY("CONFIGURACION", 25, 61);
Lcd.Linea( 0, 53, 127, 53, 1 );
Lcd.WriteXY("FECHA Y HORA", 14, 49); // Ancho, Alto, Modo
Lcd.WriteXY("TEMPERATURAS", 14, 39);
Lcd.WriteXY("FIJAR PH PARA CO2", 14, 29);
Lcd.WriteXY("HORARIO DE LUCES", 14, 19);
Lcd.WriteXY("RELES", 14, 9);
}
```

Y ahora definamos la función que controlara el teclado, esta función comprobara si pulsamos el botón OK, si lo pulsamos pintaremos la pantalla del menú con varias opciones y nos moveremos por el, si en algún momento pulsamos el botón Salir, saldremos de este menú y volveremos a la pantalla principal.

```
///////////
//
// TECLADO PRINCIPAL Y DE CONFIGURACION
//
/////////
void Teclado()
{
int Index;
boolean Salir = false;
if (digitalRead(PIN_D_KEY_OK)) //Si hemos pulsado el botón OK entramos.
{
PrintPantConfig(); // Pintamos la pantalla del menú
Index = 1;
Lcd.WriteXY( ">", 4, 59-(10*Index)); // Marca de en que posición del menú estamos
delay(1000);
//A continuación nos quedamos a la espera de que pulsemos algún botón.
while((!digitalRead(PIN_D_KEY_SALIR)) && (!Salir))
{
if (digitalRead(PIN_D_KEY_DER)) // Nos movemos para abajo
```

```

{
Lcd.WriteXY( " ", 4, 59-(10*Index));
if (Index == 5) Index = 1; else ++Index;
Lcd.WriteXY( ">", 4, 59-(10*Index));
delay(500);
}
if (digitalRead(PIN_D_KEY_IZQ)) //Nos movemos para arriba.
{
Lcd.WriteXY( " ", 4, 59-(10*Index));
if (Index == 1) Index = 5; else --Index;
Lcd.WriteXY( ">", 4, 59-(10*Index));
delay(500);
}
if (digitalRead(PIN_D_KEY_OK)) //Ejecutamos la opción del menú seleccionada.
{
switch (Index)
{
case 1:
//PrintConfReloj(); //Función para configurar el reloj
PrintPantConfig();
Lcd.WriteXY( ">", 4, 59-(10*Index));
break;
case 2:
//PrintConfTemperatura(); //Función para configurar las temperaturas
PrintPantConfig();
Lcd.WriteXY( ">", 4, 59-(10*Index));
break;
case 3:
//PrintConfPH(); //Función para configurar el PH
PrintPantConfig();
Lcd.WriteXY( ">", 4, 59-(10*Index));
break;
case 4:
//PrintConfLuces(); //Función para configurar el horario de las luces
PrintPantConfig();
Lcd.WriteXY( ">", 4, 59-(10*Index));
break;
case 5:
//PrintActivaReles(); // Función para activar y desactivar relés
PrintPantConfig();
Lcd.WriteXY( ">", 4, 59-(10*Index));
break;
}
}
}
}
PrintPantMain();
}
}
}


```

Y como es natural en nuestra función loop() pondremos una llamada a la función teclado() para que lo compruebe.

```
void loop()
```

```

{
  Lcd.WriteXY( Reloj.DateTime(), 10, 61);
  MirarTemperaturas(); //Miramos la temperaturas
  Teclado();
}

```

Ya iremos escribiendo las respectivas pantallas de configuración del menú.

Este sistema de menú y pantallas de configuración tiene una pequeña pega, que la mayoría suele obviar, y es que mientras estas dentro de un menú, el resto del código del Arduino no se esta ejecutando o lo que es lo mismo, no se ejecuta la función loop(), pero no hay pega ya que en la próxima entrega arreglaremos esta pega y algunas mas.

El código modificado del controlador lo tenéis aquí. [NuestroControlador007.zip](#)

## La función loop()

La función loop como su nombre indica es un bucle infinito que se ejecuta constantemente, lo que es muchas veces por segundo, lo único que lo retrasa es el tiempo que tarde en ejecutarse las instrucciones de código que metamos en ella, con lo cual nuestra función loop actual

```

Lcd.WriteXY( Reloj.DateTime(), 10, 61);
MirarTemperaturas(); //Miramos la temperaturas
Teclado();

```

Pintara la fecha y hora, leerá y pintara la temperatura y comprobara el teclado varias veces por segundo, lo cual es una bestialidad y que puede dar errores de lectura de la temperatura.

Para solucionar esto pondremos una forma de controlar cada que tiempo se realiza la llamada, y para ello usaremos la función millis() la cual devuelve el tiempo en milisegundos transcurridos desde que se arrancó la placa Arduino con el programa actual. Este número se desbordará (volverá a cero), después de aproximadamente 50 días.

Para empezar declararemos una variable global para guardar cuando se ejecuto la ultima vez nuestro código.

```
long Old_Millis = 0;
```

Después pondremos el siguiente código en nuestra función loop.

```

void loop()
{
  if ( (millis() - Old_Millis) > 7000 ) // comprobamos que hayan pasado 7 segundos
  {
    Old_Millis = millis(); // Guardamos el tiempo actual.
    Lcd.WriteXY( Reloj.DateTime(), 10, 61 );
    MirarTemperaturas(); //Miramos la temperaturas
  }
  Teclado();
}

```

Con este código haremos que la fecha y hora así como la lectura de la temperatura se realice cada 7 segundo, lo cual es un tiempo sobrado para que se controle todos los parámetros.  
En cambio hemos dejado fuera la función que vigila el teclado ya que esta tiene que detectar el momento exacto en que toquemos el teclado.

## La inactividad en los menús

En un post anterior pusimos el código de la función teclado() la cual nos sacaba un menú el cual se quedaba a la espera de que seleccionemos una opción, todo esto tiene un problema grave y es que mientras estamos dentro del menú el resto del código de nuestro controlador no se ejecuta, con lo cual los sensores no leen y por lo tanto tampoco actúan con nuestros actuadores, como relés.

Teniendo en cuenta esto no creáis que sea tan raro que alguien mientras está dentro de un menú se entretenga con cualquier cosa o se olvide de salir del menú.

Para evitar esto vamos a crear una función que controle la inactividad dentro de un menú y pasado de un tiempo nos saque de esta.

Empezaremos creando una variable que nos mida la inactividad dentro de un menú.

```
long Inactividad = 20000; //Tiempo que estamos sin tocar la botonera
```

Luego crearemos la función que la llamaremos EnInactividad() a la cual consultaremos para saber si llevamos más de un cierto tiempo sin tocar la botonera del menú.

```
//////////  
//  
// PARA SABER SI NOS OLVIDAMOS SALIR DE UN MENU  
//  
//////////  
boolean EnInactividad()  
{  
    if ((millis() - Inactividad) > 30000) // 30 segundos  
        return true;  
    else  
        return false;  
}
```

Luego solo tendremos que intercalarlo entre nuestro código, para ello pondremos tanto en el inicio como en cada pulsación de los botones el siguiente código

```
Inactividad = millis();
```

Con esto pondremos a cero el contador de inactividad, de esta forma si nos pasamos más de los 30 segundos configurados sin tocar ningún botón, nuestra función lo detectará y para que lo detecte la llamaremos al final del menú de esta forma.

```
if (EnInactividad())  
{  
    Pito(300);  
    Salir = true;  
}
```

De esta forma nuestro menú quedara de esta forma

```
//////////  
//  
// TECLADO PRINCIPAL Y DE CONFIGURACION  
//  
//////////  
void Teclado()  
{  
int Index;  
boolean Salir = false;  
if (digitalRead(PIN_D_KEY_OK)) //Si hemos pulsado el botón OK entramos.  
{  
PrintPantConfig(); // Pintamos la pantalla del menú  
Index = 1;  
Lcd.WriteXY( ">", 4, 59-(10*Index)); // Marca de en que posición del menú estamos  
delay(1000);  
Inactividad = millis();  
//A continuación nos quedamos a la espera de que pulsemos algún botón.  
while((!digitalRead(PIN_D_KEY_SALIR)) && (!Salir))  
{  
if (digitalRead(PIN_D_KEY_DER)) // Nos movemos para abajo  
{  
Inactividad = millis();  
Lcd.WriteXY( " ", 4, 59-(10*Index));  
if (Index == 5) Index = 1; else ++Index;  
Lcd.WriteXY( ">", 4, 59-(10*Index));  
delay(500);  
}  
if (digitalRead(PIN_D_KEY_IZQ)) //Nos movemos para arriba.  
{  
Inactividad = millis();  
Lcd.WriteXY( " ", 4, 59-(10*Index));  
if (Index == 1) Index = 5; else --Index;  
Lcd.WriteXY( ">", 4, 59-(10*Index));  
delay(500);  
}  
if (digitalRead(PIN_D_KEY_OK)) //Ejecutamos la opción del menú seleccionada.  
{  
Inactividad = millis();  
switch (Index)  
{  
case 1:  
//PrintConfReloj(); //Función para configurar el reloj  
PrintPantConfig();  
Lcd.WriteXY( ">", 4, 59-(10*Index));  
break;  
case 2:  
//PrintConfTemperatura(); //Función para configurar las temperaturas  
PrintPantConfig();  
Lcd.WriteXY( ">", 4, 59-(10*Index));  
break;  
case 3:  
//PrintConfPH(); //Función para configurar el PH
```

```

PrintPantConfig();
Lcd.WriteXY( ">", 4, 59-(10*Index));
break;
case 4:
//PrintConfLuces(); //Función para configurar el horario de las luces
PrintPantConfig();
Lcd.WriteXY( ">", 4, 59-(10*Index));
break;
case 5:
//PrintActivaReles(); // Función para activar y desactivar relés
PrintPantConfig();
Lcd.WriteXY( ">", 4, 59-(10*Index));
break;
}
}
}
if ( EnInactividad() )
{
Salir = true;
}
}
PrintPantMain();
}
}

```

## Sensores de temperatura.

Bueno y a continuación ya que estamos afinando nuestro código, vamos a afinar un poco el código de nuestros sensores de temperatura, como ya descubrió nuestro compañero **moiaio** y que después de varias pruebas tengo que darle la razón, la función setHighAlarmTemp() de la librería del sensor no es muy fina al controlar las variaciones de temperatura, por lo cual vamos a pasar de ella y controlarlo directamente.

Para empezar definiremos una constante que contenga el margen de variación de temperatura que estemos dispuestos a soportar.

```
#define MARGEN_TEMPERATURA 0.3
```

Después tendremos que borrar en la función IniciarTemperaturas() el siguiente código

```

sensors.setHighAlarmTemp( AguaTermo, 26.75 );
sensors.setLowAlarmTemp( AguaTermo, 26.25 );
sensors.setHighAlarmTemp( DepoTermo, 26.75 );
sensors.setLowAlarmTemp( DepoTermo, 26.25 );
sensors.setHighAlarmTemp( TapaTermo, 30.00 );
sensors.setLowAlarmTemp( TapaTermo, 29.50 );

```

que además tenia unos valores fijos y no tenia los valores de nuestra configuración.

Luego en la función MirarTemperaturas() cambiaremos las comprobación de temperaturas por este código

```

if ((sensors.getTempC(AguaTermo) < (CFG.TermoAcuario-MARGEN_TEMPERATURA)) &&
digitalRead(PIN_D_RELE_CALENTADOR))
SetRele( 8, PIN_D_RELE_CALENTADOR, HIGH );
if ((sensors.getTempC(AguaTermo) > (CFG.TermoAcuario+MARGEN_TEMPERATURA)) && (!
digitalRead(PIN_D_RELE_CALENTADOR)))

```

```

SetRele( 8, PIN_D_RELÉ_CALENTADOR, LOW );
if ((sensors.getTempC(DepoTermo) < (CFG.TermoDeposito-MARGEN_TEMPERATURA)) &&
digitalRead(PIN_D_RELÉ_CALENTDEPO))
SetRele( 7, PIN_D_RELÉ_CALENTDEPO, HIGH );
if ((sensors.getTempC(DepoTermo) > (CFG.TermoDeposito+MARGEN_TEMPERATURA)) &&
(!digitalRead(PIN_D_RELÉ_CALENTDEPO)))
SetRele( 7, PIN_D_RELÉ_CALENTDEPO, LOW );
if ((sensors.getTempC(TapaTermo) > (CFG.TermoTapa+MARGEN_TEMPERATURA)) &&
digitalRead(PIN_D_RELÉ_EXTRATORES))
SetRele( 6, PIN_D_RELÉ_EXTRATORES, HIGH );
if ((sensors.getTempC(TapaTermo) < (CFG.TermoTapa-MARGEN_TEMPERATURA)) && (!
digitalRead(PIN_D_RELÉ_EXTRATORES)))
SetRele( 6, PIN_D_RELÉ_EXTRATORES, LOW );

```

Como comprobareis estamos comprobando si la temperatura de nuestra configuración es superior o inferior al margen definido por nosotros, de esta forma cambiando el margen podemos controlar la precisión de control de nuestras temperaturas.

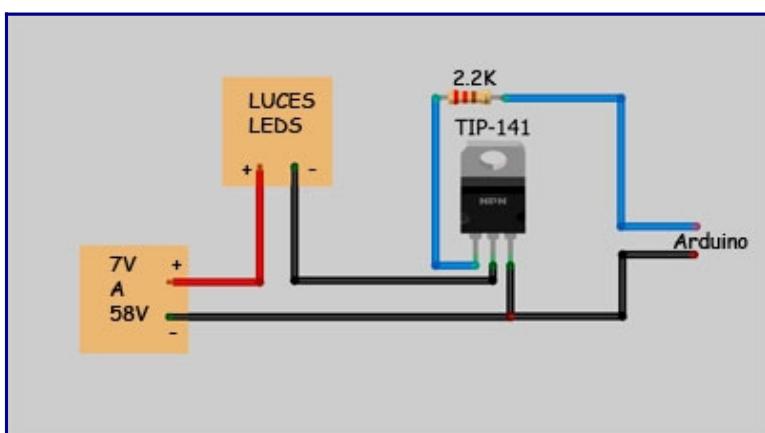
En la próxima le meteremos mano a como controlar y dimear nuestras luces.

El código modificado del controlador lo tenéis aquí. [NuestroControlador008.zip](#)

## Control y dímeo de luces.

Como ya vimos en un post anterior las luces de nuestro acuario las podemos controlar con nuestro Arduino fácilmente con tan solo un transistor y una resistencia, de una forma muy eficiente y como veremos a continuación lo podemos hacer ya sea nuestra iluminación leds o fluorescentes.

En el caso de iluminación led el sistema es simple:

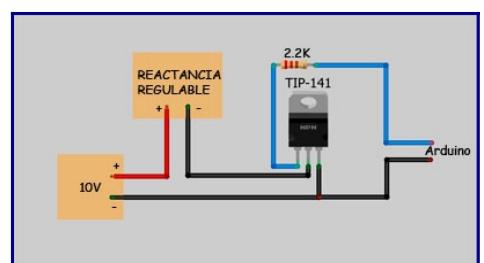


Como podemos ver el transistor TIP esta montado como si fuera una llave con la que regulamos el flujo de corriente (imaginaros que es una tubería por la que corre el agua y el TIP es la llave con la que regulamos el flujo de agua), la corriente que podemos regular puede estar entre 7v y 58v o incluso mas dependiendo del tipo de transistor, con lo cual la fuente de alimentación que regulemos puede ser de 12V, 24V o 48V dependiendo de las características de nuestra iluminación.

El control es simple, el pin del Arduino nos da de 0V a 5V escalado en 255 valores (0=0V a 255=5V) esta escala es igual para la salida de nuestro transistor el cual escalara el voltaje de la fuente de alimentación de la misma forma (ejemplo: fuente de 24V, 0=0V a 255=24V).

Si lo que queremos controlar son fluorescentes, también lo podemos hacer pero para ello nos hace falta que estén equipados con reactancias (balastros) electrónicos regulables, estas reactancias se regulan aplicándoles una señal de 0V a 10V en unas bornas para tal fin.

Lo que cambia es la fuente de alimentación que tiene que ser de 10V.



Bueno empecemos con el código para controlar nuestras luces.

Para controlar las luces debemos controlar dos cosas, el dímetro de las luces con los TIP y la fuente de alimentación con un relé, ya que aunque apaguemos las luces con los TIP, mientras no desconectemos la fuente, esta seguirá consumiendo y haciendo trabajar a los TIP, tener en cuenta que los TIP sufrirán (calentándose) más contra más reduzcamos la iluminación, ya que los TIP tienen que disparar el voltaje que recortamos en forma de calor.

El código lo haremos para controlar la iluminación en 2 faces o grupos de luces y una tercera para controlar la luz de luna.

Empezaremos definiendo el tope de dímetro de nuestras luces, en mi caso por ejemplo como me pase poniendo luces me vino muy bien para poder reducir la máxima iluminación.

```
#define DIMEO_LUZ1 255  
#define DIMEO_LUZ2 255  
#define DIMEO_NOCHE 255
```

declarando 3 variables globales que nos dirán el nivel de dímetro de nuestros nuestras luces.

```
int Dimeo1, Dimeo2, Dimeo3;
```

las iniciaremos a cero en nuestro setup()

```
Dimeo1 = 0; Dimeo2 = 0; Dimeo3 = 0;
```

Crearemos la función que pinte en la pantalla las barras de progresos de iluminación de nuestras luces.

```
//////////  
//  
// PINTA EL GAUGE CON EL NIVEL DE INTENSIDAD ACTUAL DE LA LUZ  
//  
//////////  
void GaugeXY( int Valor, byte IndexCol )  
{  
    int Col [] = {0, 109, 115, 121};  
    int X = Col[IndexCol];  
    Lcd.Rectangulo(X, 12, X+3, 51);  
    Lcd.Borrar(X+1, 12+1, X+2, 50);  
    Lcd.Rectangulo(X+1, 12, X+2, 12+(map(Valor, 0, 255, 0, 39)));  
}
```

Añadiremos la llamada a esta función en el pintado de la pantalla principal

```
//////////  
//  
// PINTA LA PANTALLA PRINCIPAL  
//  
//////////  
void PrintPantMain()  
{  
    Lcd.Cls();  
    Lcd.Rectangulo(0, 0, 127, 63);
```

```

Lcd.Linea( 0, 53, 127, 53, 1);
Lcd.Linea( 90, 10, 90, 53, 1);
Lcd.Linea( 0, 10, 127, 10, 1);
Lcd.WriteXY("ACUARIO:", 2, 50); // Ancho, Alto, Modo
Lcd.WriteXY("TAPADERA:", 2, 40);
Lcd.WriteXY("DEPOSITO:", 2, 30);
Lcd.WriteXY("PH:", 2, 20);
for(I=0; I < 8; I++)
{
Lcd.Rectangulo( 95, 50-(I*5), 104, 47-(I*5));
}
GaugeXY( Dimeo1, 1 );
GaugeXY( Dimeo2, 2 );
GaugeXY( Dimeo3, 3 );
RePintarRele();
}

```

Y crearemos la función de control y de dimeo de nuestras luces

```

///////////
// CONTROL DE LAS LUCES
/////////
void Luces()
{
// LUZ N°1 /////////////
if(Reloj.IsLater( CFG.Luz1H_ON, CFG.Luz1M_ON ) && Reloj.IsPrevious( CFG.Luz1H_OFF,
CFG.Luz1M_OFF ))
{
// Alimentar el transformador de la luz si no esta encendido
if (digitalRead(PIN_D_RELLE_LUCES)) SetRele( 5, PIN_D_RELLE_LUCES, HIGH );
if ( Dimeo1 < DIMEO_LUZ1 )
{
Dimeo1 += 2;
if (Dimeo1 > DIMEO_LUZ1) Dimeo1 = DIMEO_LUZ1;
// Si se a re-iniciado el sistema
if(Reloj.IsLater( CFG.Luz1H_ON, CFG.Luz1M_ON+30 ) &&
Reloj.IsPrevious( CFG.Luz1H_OFF, CFG.Luz1M_OFF )) Dimeo1 = DIMEO_LUZ1;
analogWrite( PIN_P_LUCES1, Dimeo1 );
}
GaugeXY( Dimeo1, 1 );
}
if( Reloj.IsLater( CFG.Luz1H_OFF, CFG.Luz1M_OFF ) && (Dimeo1 > 0) )
{
Dimeo1 -= 2;
if (Dimeo1 < 0) Dimeo1 = 0;
// Si se a re-iniciado el sistema
if (Reloj.IsLater( CFG.Luz1H_OFF, CFG.Luz1M_OFF+30 )) Dimeo1 = 0;
analogWrite( PIN_P_LUCES1, Dimeo1 );
GaugeXY( Dimeo1, 1 );
}
// LUZ N°2 /////////////

```

```

if(Reloj.IsLater( CFG.Luz2H_ON, CFG.Luz2M_ON ) && Reloj.IsPrevious( CFG.Luz2H_OFF,
CFG.Luz2M_OFF ))
{
// Alimentar el transformador de la luz si no esta encendido
if (digitalRead(PIN_D_RELE_LUCES)) SetRele( 5, PIN_D_RELE_LUCES, HIGH );
if ( Dimeo2 < DIMEO_LUZ2 )
{
Dimeo2 += 2;
if (Dimeo2 > DIMEO_LUZ2) Dimeo2 = DIMEO_LUZ2;
// Si se a re-iniciado el sistema
if(Reloj.IsLater( CFG.Luz2H_ON, CFG.Luz2M_ON+30 ) &&
Reloj.IsPrevious( CFG.Luz2H_OFF, CFG.Luz2M_OFF )) Dimeo2 = DIMEO_LUZ2;
analogWrite( PIN_P_LUCES2, Dimeo2 );
}
GaugeXY( Dimeo2, 2 );
}
if( Reloj.IsLater( CFG.Luz2H_OFF, CFG.Luz2M_OFF ) && (Dimeo2 > 0) )
{
Dimeo2 -= 2;
if (Dimeo2 < 0) Dimeo2 = 0;
// Si se a re-iniciado el sistema
if (Reloj.IsLater( CFG.Luz2H_OFF, CFG.Luz2M_OFF+30 )) Dimeo2 = 0;
analogWrite( PIN_P_LUCES2, Dimeo2 );
GaugeXY( Dimeo2, 2 );
}
// LUZ Nº3 /////////////////////////////////
if(Reloj.IsLater( CFG.Luz3H_ON, CFG.Luz3M_ON ) && Reloj.IsPrevious( CFG.Luz3H_OFF,
CFG.Luz3M_OFF ))
{
// Alimentar el transformador de la luz si no esta encendido
if (digitalRead(PIN_D_RELE_LUCES)) SetRele( 5, PIN_D_RELE_LUCES, HIGH );
if ( Dimeo3 < DIMEO_NOCHE )
{
Dimeo3 += 2;
if (Dimeo3 > DIMEO_NOCHE) Dimeo3 = DIMEO_NOCHE;
// Si se a rei-iniciado el sistema
if(Reloj.IsLater( CFG.Luz3H_ON, CFG.Luz3M_ON+30 ) &&
Reloj.IsPrevious( CFG.Luz3H_OFF, CFG.Luz3M_OFF )) Dimeo3 = DIMEO_NOCHE;
analogWrite( PIN_P_LUCES3, Dimeo3 );
}
GaugeXY( Dimeo3, 3 );
}
if( Reloj.IsLater( CFG.Luz3H_OFF, CFG.Luz3M_OFF ) && (Dimeo3 > 0) )
{
Dimeo3 -= 2;
if (Dimeo3 < 0) Dimeo3 = 0;
// Si se a rei-iniciado el sistema
if (Reloj.IsLater( CFG.Luz3H_OFF, CFG.Luz3M_OFF+30 )) Dimeo3 = 0;
analogWrite( PIN_P_LUCES3, Dimeo3 );
// Apagamos el transformador de la luz
if ( Dimeo3 == 0 ) SetRele( 5, PIN_D_RELE_LUCES, LOW );
GaugeXY( Dimeo3, 3 );
}
}

```

Y ya por ultimo ponemos la llamada a nuestra función de dimeo y control de nuestras luces en la función loop().

```
void loop()
{
if ( (millis() - Old_Millis) > 7000 ) // 7 segundos
{
Old_Millis = millis();
Lcd.WriteXY( Reloj.DateTime(), 10, 61 );
MirarTemperaturas(); //Miramos la temperaturas
Luces();
}
Teclado();
}
```

En este punto tenemos que mirar bien como funciona todo esto, por una parte tenemos 3 variables globales que en las cuales tenemos siempre el nivel de dimeo de nuestras luces y por otra parte tenemos la función Luces() que comprueba las horas de configuración de nuestras luces y dependiendo de esto va dimeando nuestras luces cada vez que la llamamos.

Como vemos en nuestra función loop() la función Luces() se llama cada 7 segundos, con lo cual llegada la hora de encender las luces, cada 7 segundos que llamemos a la función Luces() subiremos el dimeo de la luz 2 puntos de los 255 posibles, lo cual no da que desde el inicio de dimeo de una luz hasta su dimeo total (iluminación a tope) pasaran aproximadamente 15 minutos.

En mi caso que tengo configurada las primera fase 03:00 y la segunda fase a las 03:20 me queda una bonito amanecer de algo mas de media hora, lo mismo pasa la apagar las luces que se van oscureciendo gradualmente durante el mismo tiempo y que además cuando empieza el oscurecimiento de la segunda fase, también empieza el encendido gradual de la luz de luna.

Espero haberme explicado bien, de todas formas lo que no comprendáis, preguntarlo.

Nos queda todavía por ver la programación de los menús de configuración de fechas, temperaturas y horarios de luces.

El código modificado del controlador lo tenéis aquí. [NuestroControlador009.zip](#)

## Pitos y alarmas

Como ya vimos al principio del pos el controlador tiene montado un zumbador de esos pequeños, como los que traen los ordenadores.



[Mas información](#)

Como veréis este tipo de zumbador tiene 2 polos uno negativo y otro positivo y su conexión con el Arduino es muy simple ponemos el conector negativo al negativo del Arduino y el positivo a un pin digital del Arduino en este caso lo pondremos al pin 8.

Es tan simple que no creo que haga falta poner ni un esquema ni nada mas, solo el código para manejarlo.

Definimos el pin donde tenemos conectado el zumbador.

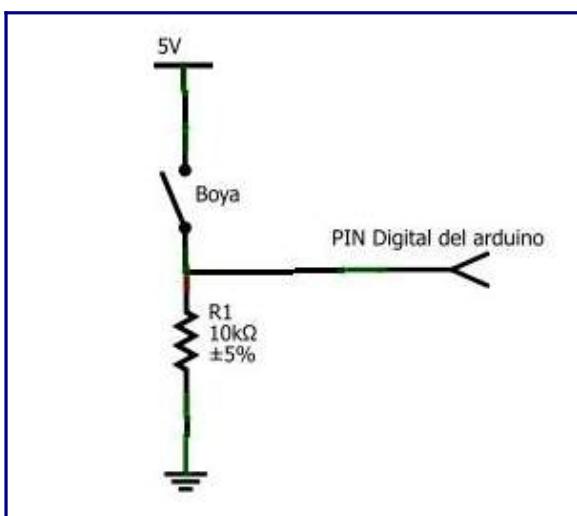
```
#define PIN_D_PITO 8
```

Y creamos una función a la que llamaremos Pito() y le pasaremos un parámetro que es la duración del pitido, cuanto mas largo sea el pitido mas fuerte lo oiremos ya que es un sonido en crescendo.

```
//////////  
//  
// GENERA UN PITIDO DE AVISO  
//  
//////////  
void Pito( int duracion )  
{  
    tone( PIN_D_PITO, 2048, duracion );  
}
```

Internamente usaremos la función del Arduino tone() la cual es una función que le tenemos que pasar el pin donde lo tenemos conectado, la frecuencia a la que queremos que suene y la duración.

Para que queremos un pito en nuestro controlador, pues os puedo asegurar que sin este, parece tonto el controlador, porque por mucho que toquemos nunca nos dice nada, por eso le pondremos que genere un pito largo cuando se reinicie para que nos enteremos que se esta reiniciando, y luego pondremos un pitido corto cada vez que le pidamos una orden y la realice, por ejemplo cuando le asignemos una configuración y la guarde.



Como he dicho al principio también tenemos que programar las alarmas y la alarma principal que tenemos que controlar es la de los niveles, si no queremos que la mujer nos aporree con la fregona.

Como ya dijimos en un post anterior el sensor de nivel (la boyta) se comporta como si fuera un pulsador y lo tenemos que conectar también como tal, con una resistencia de 10K a masa y la otra punta al positivo.

[El post en el que lo comente](#)

Luego en nuestro código será fácil implementarlo como siempre primero definiremos el pin en el que esta conectado.

```
#define PIN_D_NIVELD 6
```

Y como siempre crearemos una función que se encargue de comprobar que el sensor de nivel no se ha activado, a la cual llamaremos niveles().

```
//////////  
//  
// COMPRUEBA EL NIVEL DE AGUA DEL DEPOSITO  
//  
//////////  
void MirarNiveles()
```

```

{
if ((digitalRead(PIN_D_NIVELD) == HIGH) && (!AlarmaNivel)) // La bolla esta activa y cierra
las electroválvulas
{
SetRele( 3, PIN_D_RELÉ_OSMOSIS, LOW );
SetRele( 2, PIN_D_RELÉ_LLUVIA, LOW );
LogEvent( MSG_NIVEL, "Activada la alarma de nivel", "", "" );
MsgPie( "Alarma de nivel" );
AlarmaNivel = true;
Alarma();
}
}

```

Como veréis esta función comprueba nuestro sensor de nivel y si ve que esta activado desconecta los relés de entrada de agua como son los de ósmosis y de lluvia, y además saca un mensaje en el pie de la pantalla y llama a la función Alarma para avisarnos, esta función alarma que crearemos a continuación

```

///////////
// GENERA PITIDOS DE ALARMA
//
/////////
void Alarma()
{
int n;
for( n=0; n < 30; n++)
{
Pito(800);
delay(1000);
Pito(400);
}
delay(10000);
}

```

La cual utilizando la función Pito() nos generara un pitido repetitivo para avisarnos.

Esto tiene una pega, y es que si estará constantemente sonando la alarma hasta que no vaciemos el nivel del agua que a echo saltar la alarma de nivel.

Para evitar esto pondremos una variable global que solemos llamar semáforo, la cual nos dejara llamar a la función de comprobar nivel, siempre y cuando no nos haya avisado anteriormente de una alarma, miraremos también que si no esta la alarma activa, pongamos esta variable a false.

Definimos la variable

```
boolean AlarmaNivel = false;
```

y luego de todo esto nuestro loop() nos quedaría así.

```

void loop()
{
if ( (millis() - Old_Millis) > 7000 ) // 7 segundos
{
Old_Millis = millis();
Lcd.WriteXY( Reloj.DateTime(), 10, 61 );
if ((AlarmaNivel) && (!digitalRead(PIN_D_NIVELD))) AlarmaNivel = false;
}
}

```

```

MirarNiveles();
MirarTemperaturas(); //Miramos la temperaturas
Luces();
}
Teclado();
}

```

Bueno ya tan solo nos queda las pantallas de configuración que espero poder poner mañana.

El código modificado del controlador lo tenéis aquí. [NuestroControlador010.zip](#)

Bueno vamos a poner ahora el resto de las configuraciones, después de mucho mirarlo, no veo la forma de explicarlo, sobre todo porque casi todo esta ya explicado en la función que esplique y puse, la función Teclado().

Para la configuración que nos falta pondremos una pantalla para cada tipo de configuración y iremos llamando a estas pantallas en la función Teclado() según seleccionemos cada opción.

La función Teclado() con todas las opciones ya programadas nos quedaría asín.

```

///////////
// TECLADO PRINCIPAL Y DE CONFIGURACION
////
void Teclado()
{
int Index;
boolean Salir = false;
if (digitalRead(PIN_D_KEY_OK))
{
Pito(100);
PrintPantConfig();
Index = 1;
Lcd.WriteXY( ">", 4, 59-(10*Index));
delay(1000);
Inactividad = millis();
while(!digitalRead(PIN_D_KEY_SALIR)) && (!Salir))
{
if (digitalRead(PIN_D_KEY_DER))
{
Inactividad = millis();
Lcd.WriteXY( " ", 4, 59-(10*Index));
if (Index == 5) Index = 1; else ++Index;
Lcd.WriteXY( ">", 4, 59-(10*Index));
delay(500);
}
if (digitalRead(PIN_D_KEY_IZQ))
{
Inactividad = millis();
Lcd.WriteXY( " ", 4, 59-(10*Index));
if (Index == 1) Index = 5; else --Index;
Lcd.WriteXY( ">", 4, 59-(10*Index));
delay(500);
}
}
}

```

```

}

if (digitalRead(PIN_D_KEY_OK))
{
Inactividad = millis();
switch (Index)
{
case 1:
Pito(100);
PrintConfReloj();
PrintPantConfig();
Lcd.WriteXY( ">", 4, 59-(10*Index));
break;
case 2:
Pito(100);
PrintConfTemperatura();
PrintPantConfig();
Lcd.WriteXY( ">", 4, 59-(10*Index));
break;
case 3:
Pito(100);
PrintConfPH();
PrintPantConfig();
Lcd.WriteXY( ">", 4, 59-(10*Index));
break;
case 4:
Pito(100);
PrintConfLuces();
PrintPantConfig();
Lcd.WriteXY( ">", 4, 59-(10*Index));
break;
case 5:
Pito(100);
PrintActivaReles();
PrintPantConfig();
Lcd.WriteXY( ">", 4, 59-(10*Index));
break;
}
}
if ( EnInactividad() )
{
Pito(300);
Salir = true;
}
}
PrintPantMain();
}
}

```

y la programación de las pantallas de configuración.

```
//////////  
//  
// CONFIGURACION RELOJ  
//  
//////////  
void PrintConfReloj()  
{  
int Index, IndexCol;  
int Col[] = {2,51};  
int monthDay, month, year, hour, minute, second;  
static char tmp[25];  
boolean Salir = false;  
  
Reloj.GetDateTime( monthDay, month, year, hour, minute, second );  
  
Lcd.Cls();  
Lcd.Rectangulo(0, 0, 127, 63);  
Lcd.WriteXY("RELOJ", 40, 61);  
Lcd.Linea( 0, 53, 127, 53, 1 );  
Lcd.WriteXY("DIA:", 8, 49); Lcd.WriteXY("HORA :", 58, 49);  
Lcd.WriteXY("MES:", 8, 39); Lcd.WriteXY("MINUTOS :", 58, 39);  
Lcd.WriteXY("ANO:", 8, 29); Lcd.WriteXY("SEGUNDOS:", 58, 29);  
Lcd.WriteXY( ">", 2, 49 );  
Index = 1;  
IndexCol = 0;  
  
sprintf(tmp, "%02d", monthDay); Lcd.WriteXY(tmp, 34, 49);  
sprintf(tmp, "%02d", hour); Lcd.WriteXY(tmp, 112, 49);  
sprintf(tmp, "%02d", month); Lcd.WriteXY(tmp, 34, 39);  
sprintf(tmp, "%02d", minute); Lcd.WriteXY(tmp, 112, 39);  
sprintf(tmp, "%02d", year); Lcd.WriteXY(tmp, 34, 29);  
sprintf(tmp, "%02d", second); Lcd.WriteXY(tmp, 112, 29);  
  
Inactividad = millis();  
while(!digitalRead(PIN_D_KEY_SALIR)) && (!Salir)  
{  
if (digitalRead(PIN_D_KEY_DER)) // Adelante  
{  
Inactividad = millis();  
Lcd.WriteXY( " ", Col[IndexCol], 59-(10*Index) );  
if (Index == 3)  
{  
Index = 1;  
if (IndexCol == 1) IndexCol = 0; else IndexCol = 1;  
}  
else ++Index;  
Lcd.WriteXY( ">", Col[IndexCol], 59-(10*Index) );  
delay(1000);  
}  
if (digitalRead(PIN_D_KEY_IZQ)) // Atras  
{  
Inactividad = millis();
```

```

Lcd.WriteXY( " ", Col[IndexCol], 59-(10*Index) );
if (Index == 1)
{
Index = 3;
if (IndexCol == 1) IndexCol = 0; else IndexCol = 1;
}
else --Index;
Lcd.WriteXY( ">", Col[IndexCol], 59-(10*Index) );
delay(1000);
}
if (digitalRead(PIN_D_KEY_MAS))
{
Inactividad = millis();
switch (Index+(IndexCol*10))
{
case 1:
monthDay++;
if (monthDay > 31) monthDay = 1;
sprintf(tmp, "%02d", monthDay);
Lcd.WriteXY(tmp, 34, 49);
break;
case 2:
month++;
if (month > 12) month = 1;
sprintf(tmp, "%02d", month);
Lcd.WriteXY(tmp, 34, 39);
break;
case 3:
year++;
Lcd.WriteXY(tmp, 34, 29);
sprintf(tmp, "%02d", year);
break;
case 11:
hour++;
if (hour > 24) hour = 1;
sprintf(tmp, "%02d", hour);
Lcd.WriteXY(tmp, 112, 49);
break;
case 12:
minute++;
if (minute > 59) minute = 0;
sprintf(tmp, "%02d", minute);
Lcd.WriteXY(tmp, 112, 39);
break;
case 13:
second++;
if (second > 59) second = 0;
sprintf(tmp, "%02d", second);
Lcd.WriteXY(tmp, 112, 29);
break;
}
}
if (digitalRead(PIN_D_KEY_MENOS))
{

```

```

Inactividad = millis();
switch (Index+(IndexCol*10))
{
    case 1:
        monthDay--;
        if (monthDay < 1) monthDay = 31;
        sprintf(tmp, "%02d", monthDay);
        Lcd.WriteXY(tmp, 34, 49);
        break;
    case 2:
        month--;
        if (month < 1) month = 12;
        sprintf(tmp, "%02d", month);
        Lcd.WriteXY(tmp, 34, 39);
        break;
    case 3:
        year-- ;
        sprintf(tmp, "%02d", year);
        Lcd.WriteXY(tmp, 34, 29);
        break;
    case 11:
        hour-- ;
        if (hour < 0) hour = 24;
        sprintf(tmp, "%02d", hour);
        Lcd.WriteXY(tmp, 112, 49);
        break;
    case 12:
        minute-- ;
        if (minute < 0) minute = 59;
        sprintf(tmp, "%02d", minute);
        Lcd.WriteXY(tmp, 112, 39);
        break;
    case 13:
        second-- ;
        if (second < 0) second = 59;
        sprintf(tmp, "%02d", second);
        Lcd.WriteXY(tmp, 112, 29);
        break;
    }
}
if (digitalRead(PIN_D_KEY_OK))
{
    Reloj.SetDateTime( monthDay, month, year, hour, minute, second );
    Pito(100);
    Salir = true;
}
Lcd.WriteXY( Reloj.Date(), 10, 10);
Lcd.WriteXY( Reloj.Time(), 70, 10);
if ( EnInactividad() )
{
    Pito(300);
    Salir = true;
}
}

```

```

}

///////////////////////////////
// CONFIGURACION TEMPERATURA
//
/////////////////////////////
void PrintConfTemperatura()
{
int Index;
float Temp1, Temp2, Temp3;
boolean Salir = false;

Lcd.Cls();
Lcd.Rectangulo(0, 0, 127, 63);
Lcd.WriteXY("TEMPERATURA", 25, 61);
Lcd.Linea( 0, 53, 127, 53, 1 );
Lcd.WriteXY("ACUARIO :", 8, 49);
Lcd.WriteXY("TAPA :", 8, 39);
Lcd.WriteXY("DEPOSITO:", 8, 29);
Lcd.WriteXY( ">", 2, 49 );
Index = 1;

Temp1 = CFG.TermoAcuario;
Temp2 = CFG.TermoTapa;
Temp3 = CFG.TermoDeposito;

Lcd.WriteXY(Temp1, 65, 49, 1);
Lcd.WriteXY(Temp2, 65, 39, 1);
Lcd.WriteXY(Temp3, 65, 29, 1);

Inactividad = millis();
while((!digitalRead(PIN_D_KEY_SALIR)) && (!Salir))
{
if (digitalRead(PIN_D_KEY_DER)) // Adelante
{
Inactividad = millis();
Lcd.WriteXY( " ", 2, 59-(10*Index) );
if (Index == 3)
{
Index = 1;
}
else ++Index;
Lcd.WriteXY( ">", 2, 59-(10*Index) );
delay(1000);
}
if (digitalRead(PIN_D_KEY_IZQ)) // Atras
{
Inactividad = millis();
Lcd.WriteXY( " ", 2, 59-(10*Index) );
if (Index == 1)
{
Index = 3;
}
}
}
}

```

```

else --Index;
Lcd.WriteXY( ">", 2, 59-(10*Index) );
delay(1000);
}
if (digitalRead(PIN_D_KEY_MAS))
{
Inactividad = millis();
switch (Index)
{
case 1:
Temp1 += 0.5;
Lcd.WriteXY(Temp1, 65, 49, 1);
break;
case 2:
Temp2 += 0.5;
Lcd.WriteXY(Temp2, 65, 39, 1);
break;
case 3:
Temp3 += 0.5;
Lcd.WriteXY(Temp3, 65, 29, 1);
break;
}
}
if (digitalRead(PIN_D_KEY_MENOS))
{
Inactividad = millis();
switch (Index)
{
case 1:
Temp1 -= 0.5;
Lcd.WriteXY(Temp1, 65, 49, 1);
break;
case 2:
Temp2 -= 0.5;
Lcd.WriteXY(Temp2, 65, 39, 1);
break;
case 3:
Temp3 -= 0.5;
Lcd.WriteXY(Temp3, 65, 29, 1);
break;
}
}
if (digitalRead(PIN_D_KEY_OK))
{
sensors.setHighAlarmTemp( AguaTermo, Temp1 + MARGEN_TEMPERATURA );
sensors.setLowAlarmTemp( AguaTermo, Temp1 - MARGEN_TEMPERATURA );
sensors.setHighAlarmTemp( DepoTermo, Temp3 + MARGEN_TEMPERATURA );
sensors.setLowAlarmTemp( DepoTermo, Temp3 - MARGEN_TEMPERATURA );
sensors.setHighAlarmTemp( TapaTermo, Temp2 );
sensors.setLowAlarmTemp( TapaTermo, Temp2 );
CFG.TermoAcuario = Temp1;
CFG.TermoTapa = Temp2;
CFG.TermoDeposito = Temp3;
SetConfig();
}

```

```

Pito(100);
Salir = true;
}
if ( EnInactividad() )
{
Pito(300);
Salir = true;
}
}

///////////
// CONFIGURACION DEL PH
//
/////////
void PrintConfPH()
{
boolean Salir = false;
Lcd.Cls();
Lcd.Rectangulo(0, 0, 127, 63);
Lcd.WriteXY("PH", 60, 61);
Lcd.Linea( 0, 53, 127, 53, 1 );
Lcd.WriteXY("PH :", 8, 39);

Lcd.WriteXY(CFG.PH, 55, 39, 1);

Inactividad = millis();
while((!digitalRead(PIN_D_KEY_SALIR)) && (!Salir))
{
if (digitalRead(PIN_D_KEY_MAS))
{
Inactividad = millis();
CFG.PH += 0.1 ;
Lcd.WriteXY(CFG.PH, 55, 39, 1);
}
if (digitalRead(PIN_D_KEY_MENOS))
{
Inactividad = millis();
CFG.PH -= 0.1 ;
Lcd.WriteXY(CFG.PH, 55, 39, 1);
}
if (digitalRead(PIN_D_KEY_OK))
{
SetConfig();
Pito(100);
Salir = true;
}
if ( EnInactividad() )
{
Pito(300);
Salir = true;
}
}
}

```

```

}

////////// CONFIGURACION DE LOS HORARIOS DE LAS LUCES
//////////
void PrintConfLuces()
{
    int IndexLine, IndexCol;
    int Line[] = {39,22,5};
    int Col[] = {42,59,90,107};
    int MarcaLine[] = {41,24,7};
    static char tmp[25];
    boolean Salir = false;

    Lcd.Cls();
    Lcd.Rectangulo(0, 0, 127, 63);
    Lcd.WriteXY("LUCES", 50, 61);
    Lcd.Linea( 0, 53, 127, 53, 1 );
    Lcd.WriteXY("FASE1:", 3, 49); Lcd.WriteXY(":", 53, 49); Lcd.WriteXY("a", 77, 49);
    Lcd.WriteXY(":", 101, 49);
    Lcd.WriteXY("FASE2:", 3, 32); Lcd.WriteXY(":", 53, 32); Lcd.WriteXY("a", 77, 32);
    Lcd.WriteXY(":", 101, 32);
    Lcd.WriteXY("LUNA :", 3, 15); Lcd.WriteXY(":", 53, 15); Lcd.WriteXY("a", 77, 15);
    Lcd.WriteXY(":", 101, 15);
    IndexLine = 0;
    IndexCol = 0;

    sprintf(tmp, "%02d", CFG.Luz1H_ON); Lcd.WriteXY(tmp, 42, 49); sprintf(tmp, "%02d",
    CFG.Luz1M_ON); Lcd.WriteXY(tmp, 59, 49);
    sprintf(tmp, "%02d", CFG.Luz2H_ON); Lcd.WriteXY(tmp, 42, 32); sprintf(tmp, "%02d",
    CFG.Luz2M_ON); Lcd.WriteXY(tmp, 59, 32);
    sprintf(tmp, "%02d", CFG.Luz3H_ON); Lcd.WriteXY(tmp, 42, 15); sprintf(tmp, "%02d",
    CFG.Luz3M_ON); Lcd.WriteXY(tmp, 59, 15);

    sprintf(tmp, "%02d", CFG.Luz1H_OFF); Lcd.WriteXY(tmp, 90, 49); sprintf(tmp, "%02d",
    CFG.Luz1M_OFF); Lcd.WriteXY(tmp, 107, 49);
    sprintf(tmp, "%02d", CFG.Luz2H_OFF); Lcd.WriteXY(tmp, 90, 32); sprintf(tmp, "%02d",
    CFG.Luz2M_OFF); Lcd.WriteXY(tmp, 107, 32);
    sprintf(tmp, "%02d", CFG.Luz3H_OFF); Lcd.WriteXY(tmp, 90, 15); sprintf(tmp, "%02d",
    CFG.Luz3M_OFF); Lcd.WriteXY(tmp, 107, 15);
    Lcd.WriteXY("^^", 42, 41);

    Inactividad = millis();
    while(!digitalRead(PIN_D_KEY_SALIR)) && (!Salir)
    {
        if (digitalRead(PIN_D_KEY_DER)) // Adelante
        {
            Inactividad = millis();
            Lcd.WriteXY(" ", Col[IndexCol], MarcaLine[IndexLine] );
            if (IndexCol == 3)
            {
                IndexCol = 0;

```

```

if (IndexLine == 2) IndexLine = 0; else IndexLine++;
}
else ++IndexCol;
Lcd.WriteXY( "^^", Col[IndexCol], MarcaLine[IndexLine] );
delay(100);
}
if (digitalRead(PIN_D_KEY_IZQ)) // Atras
{
Inactividad = millis();
Lcd.WriteXY( " ", Col[IndexCol], MarcaLine[IndexLine] );
if (IndexCol == 0)
{
IndexCol = 3;
if (IndexLine == 0) IndexLine = 2; else IndexLine--;
}
else --IndexCol;
Lcd.WriteXY( "^^", Col[IndexCol], MarcaLine[IndexLine] );
delay(100);
}

if (digitalRead(PIN_D_KEY_MAS))
{
Inactividad = millis();
switch (IndexLine+(IndexCol*10))
{
case 0:
CFG.Luz1H_ON++;
if (CFG.Luz1H_ON > 24) CFG.Luz1H_ON = 0;
sprintf(tmp, "%02d", CFG.Luz1H_ON);
Lcd.WriteXY(tmp, 42, 49);
break;
case 10:
CFG.Luz1M_ON++;
if (CFG.Luz1M_ON > 59) CFG.Luz1M_ON = 0;
sprintf(tmp, "%02d", CFG.Luz1M_ON);
Lcd.WriteXY(tmp, 59, 49);
break;
case 20:
CFG.Luz1H_OFF++;
if (CFG.Luz1H_OFF > 28) CFG.Luz1H_OFF = 0;
sprintf(tmp, "%02d", CFG.Luz1H_OFF);
Lcd.WriteXY(tmp, 90, 49);
break;
case 30:
CFG.Luz1M_OFF++;
if (CFG.Luz1M_OFF > 59) CFG.Luz1M_OFF = 0;
sprintf(tmp, "%02d", CFG.Luz1M_OFF);
Lcd.WriteXY(tmp, 107, 49);
break;
case 1:
CFG.Luz2H_ON++;
if (CFG.Luz2H_ON > 24) CFG.Luz2H_ON = 0;
sprintf(tmp, "%02d", CFG.Luz2H_ON);
Lcd.WriteXY(tmp, 42, 32);
}
}

```

```

break;
case 11:
CFG.Luz2M_ON++;
if (CFG.Luz2M_ON > 59) CFG.Luz2M_ON = 0;
sprintf(tmp, "%02d", CFG.Luz2M_ON);
Lcd.WriteXY(tmp, 59, 32);
break;
case 21:
CFG.Luz2H_OFF++;
if (CFG.Luz2H_OFF > 28) CFG.Luz2H_OFF = 0;
sprintf(tmp, "%02d", CFG.Luz2H_OFF);
Lcd.WriteXY(tmp, 90, 32);
break;
case 31:
CFG.Luz2M_OFF++;
if (CFG.Luz2M_OFF > 59) CFG.Luz2M_OFF = 0;
sprintf(tmp, "%02d", CFG.Luz2M_OFF);
Lcd.WriteXY(tmp, 107, 32);
break;
case 2:
CFG.Luz3H_ON++;
if (CFG.Luz3H_ON > 24) CFG.Luz3H_ON = 0;
sprintf(tmp, "%02d", CFG.Luz3H_ON);
Lcd.WriteXY(tmp, 42, 15);
break;
case 12:
CFG.Luz3M_ON++;
if (CFG.Luz3M_ON > 59) CFG.Luz3M_ON = 0;
sprintf(tmp, "%02d", CFG.Luz3M_ON);
Lcd.WriteXY(tmp, 59, 15);
break;
case 22:
CFG.Luz3H_OFF++;
if (CFG.Luz3H_OFF > 28) CFG.Luz3H_OFF = 0;
sprintf(tmp, "%02d", CFG.Luz3H_OFF);
Lcd.WriteXY(tmp, 90, 15);
break;
case 32:
CFG.Luz3M_OFF++;
if (CFG.Luz3M_OFF > 59) CFG.Luz3M_OFF = 0;
sprintf(tmp, "%02d", CFG.Luz3M_OFF);
Lcd.WriteXY(tmp, 107, 15);
break;
}
}
if (digitalRead(PIN_D_KEY_MENOS))
{
Inactividad = millis();
switch (IndexLine+(IndexCol*10))
{
case 0:
CFG.Luz1H_ON--;
if (CFG.Luz1H_ON < 0) CFG.Luz1H_ON = 24;
sprintf(tmp, "%02d", CFG.Luz1H_ON);

```

```

    Lcd.WriteXY(tmp, 42, 49);
break;
case 10:
    CFG.Luz1M_ON-- ;
    if (CFG.Luz1M_ON < 0) CFG.Luz1M_ON = 59;
    sprintf(tmp, "%02d", CFG.Luz1M_ON);
    Lcd.WriteXY(tmp, 59, 49);
break;
case 20:
    CFG.Luz1H_OFF-- ;
    if (CFG.Luz1H_OFF < 0) CFG.Luz1H_OFF = 28;
    sprintf(tmp, "%02d", CFG.Luz1H_OFF);
    Lcd.WriteXY(tmp, 90, 49);
break;
case 30:
    CFG.Luz1M_OFF-- ;
    if (CFG.Luz1M_OFF < 0) CFG.Luz1M_OFF = 59;
    sprintf(tmp, "%02d", CFG.Luz1M_OFF);
    Lcd.WriteXY(tmp, 107, 49);
break;
case 1:
    CFG.Luz2H_ON-- ;
    if (CFG.Luz2H_ON < 0) CFG.Luz2H_ON = 24;
    sprintf(tmp, "%02d", CFG.Luz2H_ON);
    Lcd.WriteXY(tmp, 42, 32);
break;
case 11:
    CFG.Luz2M_ON-- ;
    if (CFG.Luz2M_ON < 0) CFG.Luz2M_ON = 59;
    sprintf(tmp, "%02d", CFG.Luz2M_ON);
    Lcd.WriteXY(tmp, 59, 32);
break;
case 21:
    CFG.Luz2H_OFF-- ;
    if (CFG.Luz2H_OFF < 0) CFG.Luz2H_OFF = 28;
    sprintf(tmp, "%02d", CFG.Luz2H_OFF);
    Lcd.WriteXY(tmp, 90, 32);
break;
case 31:
    CFG.Luz2M_OFF-- ;
    if (CFG.Luz2M_OFF < 0) CFG.Luz2M_OFF = 59;
    sprintf(tmp, "%02d", CFG.Luz2M_OFF);
    Lcd.WriteXY(tmp, 107, 32);
break;
case 2:
    CFG.Luz3H_ON-- ;
    if (CFG.Luz3H_ON < 0) CFG.Luz3H_ON = 24;
    sprintf(tmp, "%02d", CFG.Luz3H_ON);
    Lcd.WriteXY(tmp, 42, 15);
break;
case 12:
    CFG.Luz3M_ON-- ;
    if (CFG.Luz3M_ON < 0) CFG.Luz3M_ON = 59;
    sprintf(tmp, "%02d", CFG.Luz3M_ON);

```

```

    Lcd.WriteXY(tmp, 59, 15);
break;
case 22:
CFG.Luz3H_OFF-- ;
if (CFG.Luz3H_OFF < 0) CFG.Luz3H_OFF = 28;
sprintf(tmp, "%02d", CFG.Luz3H_OFF);
Lcd.WriteXY(tmp, 90, 15);
break;
case 32:
CFG.Luz3M_OFF-- ;
if (CFG.Luz3M_OFF < 0) CFG.Luz3M_OFF = 59;
sprintf(tmp, "%02d", CFG.Luz3M_OFF);
Lcd.WriteXY(tmp, 107, 15);
break;
}

}

if (digitalRead(PIN_D_KEY_OK))
{
SetConfig();
Pito(100);
Salir = true;
}
if ( EnInactividad() )
{
Pito(300);
Salir = true;
}
}
}

///////////////////////////////////////////////////////////////////
//
// PANTALLAS DE ACTIVACION DE RELES
//
///////////////////////////////////////////////////////////////////
void PrintActivaReles()
{
int Index, IndexCol;
int Col[] = {2,61};
static char tmp[25];
boolean Salir = false;

Lcd.Cls();
Lcd.Rectangulo(0, 0, 127, 63);
Lcd.WriteXY("RELES", 50, 61);
Lcd.Linea( 0, 53, 127, 53, 1 );
Lcd.WriteXY("CAL1:", 8, 49); Lcd.WriteXY("CO2:", 68, 49);
Lcd.WriteXY("CAL2:", 8, 39); Lcd.WriteXY("OSMO:", 68, 39);
Lcd.WriteXY("EXTR:", 8, 29); Lcd.WriteXY("LLUV:", 68, 29);
Lcd.WriteXY("LUZ :", 8, 19); Lcd.WriteXY("LLEN:", 68, 19);
Lcd.WriteXY("FAS1:", 8, 9); Lcd.WriteXY("FAS2:", 68, 9);
Lcd.WriteXY( ">", 2, 49 );
Index = 1;
}

```

```

IndexCol = 0;

Lcd.WriteXY(OFFON(digitalRead(PIN_D_RELÉ_CALENTADOR)), 42, 49);
Lcd.WriteXY(OFFON(digitalRead(PIN_D_RELÉ_CALENTDEPO)), 42, 39);
Lcd.WriteXY(OFFON(digitalRead(PIN_D_RELÉ_EXTRATORES)), 42, 29);
Lcd.WriteXY(OFFON(digitalRead(PIN_D_RELÉ_LUCES)), 42, 19);
Lcd.WriteXY(OFFON(!IsLuz1), 42, 9);
Lcd.WriteXY(OFFON(digitalRead(PIN_D_RELÉ_CO2)), 102, 49);
Lcd.WriteXY(OFFON(digitalRead(PIN_D_RELÉ_OSMOSIS)), 102, 39);
Lcd.WriteXY(OFFON(digitalRead(PIN_D_RELÉ_LLUVIA)), 102, 29);
Lcd.WriteXY(OFFON(digitalRead(PIN_D_RELÉ_LLENADO)), 102, 19);
Lcd.WriteXY(OFFON(!IsLuz2), 102, 9);

Inactividad = millis();
while((!digitalRead(PIN_D_KEY_SALIR)) && (!Salir))
{
if (digitalRead(PIN_D_KEY_DER)) // Adelante
{
Inactividad = millis();
Lcd.WriteXY( " ", Col[IndexCol], 59-(10*Index) );
if (Index == 5)
{
Index = 1;
if (IndexCol == 1) IndexCol = 0; else IndexCol = 1;
}
else ++Index;
Lcd.WriteXY( ">", Col[IndexCol], 59-(10*Index) );
delay(1000);
}
if (digitalRead(PIN_D_KEY_IZQ)) // Atras
{
Inactividad = millis();
Lcd.WriteXY( " ", Col[IndexCol], 59-(10*Index) );
if (Index == 1)
{
Index = 5;
if (IndexCol == 1) IndexCol = 0; else IndexCol = 1;
}
else --Index;
Lcd.WriteXY( ">", Col[IndexCol], 59-(10*Index) );
delay(1000);
}
if (digitalRead(PIN_D_KEY_OK))
{
delay(50);
switch (Index+(IndexCol*10))
{
case 1:
SetRele( 8, PIN_D_RELÉ_CALENTADOR, digitalRead(PIN_D_RELÉ_CALENTADOR) );
Lcd.WriteXY(OFFON(digitalRead(PIN_D_RELÉ_CALENTADOR)), 42, 49);
break;
case 2:
SetRele( 7, PIN_D_RELÉ_CALENTDEPO, digitalRead(PIN_D_RELÉ_CALENTDEPO) );
Lcd.WriteXY(OFFON(digitalRead(PIN_D_RELÉ_CALENTDEPO)), 42, 39);
}
}
}

```

```

break;
case 3:
SetRele( 6, PIN_D_RELÉ_EXTRATORES, digitalRead(PIN_D_RELÉ_EXTRATORES) );
Lcd.WriteXY(OFFON(digitalRead(PIN_D_RELÉ_EXTRATORES)), 42, 29);
break;
case 4:
SetRele( 5, PIN_D_RELÉ_LUCES, digitalRead(PIN_D_RELÉ_LUCES) );
Lcd.WriteXY(OFFON(digitalRead(PIN_D_RELÉ_LUCES)), 42, 19);
break;
case 5:
if (IsLuz1)
{analogWrite( PIN_P_LUCES1, 0 ); IsLuz1 = false;}
else
{analogWrite( PIN_P_LUCES1, 255 ); IsLuz1 = true;}
Lcd.WriteXY(OFFON(!IsLuz1), 42, 9);
break;
case 11:
SetRele( 4, PIN_D_RELÉ_CO2, digitalRead(PIN_D_RELÉ_CO2) );
Lcd.WriteXY(OFFON(digitalRead(PIN_D_RELÉ_CO2)), 102, 49);
break;
case 12:
SetRele( 3, PIN_D_RELÉ_OSMOSIS, digitalRead(PIN_D_RELÉ_OSMOSIS) );
Lcd.WriteXY(OFFON(digitalRead(PIN_D_RELÉ_OSMOSIS)), 102, 39);
break;
case 13:
SetRele( 2, PIN_D_RELÉ_LLUVIA, digitalRead(PIN_D_RELÉ_LLUVIA) );
Lcd.WriteXY(OFFON(digitalRead(PIN_D_RELÉ_LLUVIA)), 102, 29);
break;
case 14:
SetRele( 1, PIN_D_RELÉ_LLENADO, digitalRead(PIN_D_RELÉ_LLENADO) );
Lcd.WriteXY(OFFON(digitalRead(PIN_D_RELÉ_LLENADO)), 102, 19);
break;
case 15:
if (IsLuz2)
{analogWrite( PIN_P_LUCES2, 0 ); IsLuz2 = false;}
else
{analogWrite( PIN_P_LUCES2, 255 ); IsLuz2 = true;}
Lcd.WriteXY(OFFON(!IsLuz2), 102, 9);
break;
}
}
if ( EnInactividad() )
{
Pito(300);
Salir = true;
}
}
}

///////////////////////////////
// DEVUELVE ON o OFF
//
/////////////////////////////

```

```

char * OFFON( int Valor )
{
if (Valor == LOW)
{return "ON";}
else
{return "OFF";}
}

```

Hay que tener en cuenta que el horario de las luces, tiene su propio sistema de codificación, a causa de que muchos de nosotros tendremos configuradas las luces hasta una hora superior a las 24:00, por ejemplo hasta la 01:00 o mas tarde y esto daría error al comparar si es mas tarde o mas temprano de la hora fijada. las horas de apagado superiores a las 24:00 las pasaremos sumándolas osea

Si es a las 01:00 pondremos a las 25:00

Si es a las 01:30 pondremos a las 25:30

Si es a las 02:00 pondremos a las 26:00

y así el resto de los valores, pero recordar, solo para configurar el apagado.

Lo que no comprendais preguntarlo.

El código modificado del controlador lo tenéis aquí. [NuestroControlador011.zip](#)

## Leer el PH

Vamos a ver ahora un tema que se me había olvidado, nuestro sensor de PH, como ya vimos en el post [#16](#), podemos utilizar un pequeño sensor que conectándole la sonda PH tendremos el control del PH de nuestro acuario, hay además de esta sonda otras de distinto tipo, mejores y peores, según el gusto o la dificultad de cada uno, yo me decidí por la que describo porque en su momento me pareció la mas fiable y fácil, pero para el que quiera otras opciones os pongo algunos enlaces.

[http://www.phidgets.com/products.php?product\\_id=1130](http://www.phidgets.com/products.php?product_id=1130)

<http://code.google.com/p/phduino/>

<http://www.ebay.com/itm/Arduino-pH-...d-/110813625526>

<http://www.ebay.com/itm/Arduino-pH-...d-/110813625526>

Bueno empecemos, la programación de nuestro sensor, para ello empezaremos declarando el margen que estamos dispuestos que varié nuestro PH

```
#define MARGEN_PH 0.1
```

Luego definiremos una variable global para el PH

```
float pH;
```

y luego escribimos una función que nos lea el PH, lo pinte en pantalla y controle si hay que activar el CO2 o lo que deseemos según el valor del PH, en esta estamos controlando el relé de la válvula del CO2.

Este tipo de sensor se comunica con el Arduino a través de un puerto serie, por lo que en la función deberemos tener en cuenta por que puerto serie lo tenemos conectado, además este sensor solo da lo que se le pide, o lo que es lo mismo para que nos mande el valor del PH por el puerto serie tenemos primero que pedírselo con una orden mandada por el mismo puerto serie, en este caso la orden es mandarle el valor de la temperatura actual del agua ya que es un valor que le hace falta para la escala del PH.

En la función el puerto definido para la sonda es el Serial3.

```

///////////
// FUNCION DE LECTURA DEL PH
//
///////////

float LeerPH(void)
{
int cl_i, i; // Contador
float Tem;
char stamp_data[15];
char * strtmp = " ";

pH = 0;

Tem = sensors.getTempC(AguaTermo); // La temperatura en celcius para el agua
strtmp = dtostrf( Tem, 4, 2, strtmp );
Serial3.print(strtmp); // Le mando la temperatura
Serial3.print("\r");
delay(900);
for (int x; x < 15; x++)
{
stamp_data[0] = 0;
}
cl_i = 0;
while(Serial3.available())
{
stamp_data[cl_i] = Serial3.read();
cl_i++;
}
stamp_data[cl_i]='\0';

if (stamp_data != "")
{
pH = atof( stamp_data ); // Combertimos la cadena de texto a float.

if ((pH > 1.0) && (pH < 10.0)) // Si es un float valido
{
Lcd.WriteXY( pH, 63, 20, 2 );
}
else
{
Lcd.WriteXY( "MAL ", 63, 20 );
}
}

if ((pH < (CFG.PH - MARGEN_PH)) && (!digitalRead(PIN_D_RELE_CO2)))
{
SetRele( 4, PIN_D_RELE_CO2, LOW );
}
if ((pH > (CFG.PH + MARGEN_PH)) && digitalRead(PIN_D_RELE_CO2))
{
SetRele( 4, PIN_D_RELE_CO2, HIGH );
}
}

```

```

    return pH;
}

```

Luego tan solo nos queda incluir la llamada a nuestra función en el loop()

```

void loop()
{
if ( (millis() - Old_Millis) > 7000 ) // 7 segundos
{
Old_Millis = millis();
Lcd.WriteXY( Reloj.DateTime(), 10, 61 );
if ((AlarmaNivel) && (!digitalRead(PIN_D_NIVELD))) AlarmaNivel = false;
MirarNiveles();
MirarTemperaturas(); //Miramos la temperaturas
LeerPH();
Luces();
if (!Iniciado)
{
IsLuz1 = (Dimeo1 > 0); IsLuz2 = (Dimeo2 > 0); IsLuz3 = (Dimeo3 > 0); Iniciado = true;
}
}
Teclado();
}

```

Con esto tendremos nuestro sensor de PH funcionando, pero nos falta una cosa muy importante “Calibrarlo” como todos los sensores de PH, para ello nos hace falta un par de botes de tapón de PH4 y PH7, podemos usar también uno de PH10.

El sistema es simple y fácil, meteremos la sonda de PH primero en el bote de tapón de PH4 teniendo en cuenta que tiene que estar a 25 grados de temperatura, para lo cual ayuda meter también en el bote una de las sondas de temperatura.

Después de esto tan solo tenemos que mandarle la orden de ajuste de PH4 por el puerto serie del sensor, la orden en este caso es “F”, para el PH7 “S” y para el PH “T”.

Al mandarle la orden el sensor se autocalibrara al PH dado, y con esto ya tendremos calibrado y funcionando nuestro sensor de PH.

Lo malo es que en todo esto falla una cosa, “como mandamos la orden por el puerto del sensor”, podríamos hacer un pequeño código que al pulsar un botón se mandara, pero seria complicado y muy lioso de configurar, por lo que haremos otra cosa, le daremos capacidad a nuestro Arduino para recibir ordenes desde nuestro PC, como? Pues desde el monitor del puerto serie del IDE del Arduino.

Cuando tenemos conectado nuestro Arduino al PC con el cable de USB, abrimos el IDE y le damos al icono del monitor, con esto se nos abrirá una ventana como un editor de texto el la cual nos saldrá todas las ordenes de salida del Arduino al puerto serie por defecto el 0.

Por ejemplo si ponemos en el código de nuestro Arduino la orden Serial.println('Hola'); nos saldrá “Hola” en la ventana del monitor y si lo que queremos es mandarle algo al Arduino en la parte de arriba hay una línea de edición con un botón que pone mandar, para mandar algo al Arduino tan solo tenemos que escribirlo en esa línea y darle al botón.

Claro que para todo esto nuestro Arduino tiene que tener programado un código que reciba estas ordenes y las entienda, conque vamos a programar el código para ello.

Lo primero que nos hace falta es un código que se encargue de recoger los caracteres recibidos desde el PC y nos lo devuelva sea una orden completa, para ello en vez de escribir ningún código, usaremos uno ya escrito y que esta muy bien realizado, el código es cogido del foro en inglés del Arduino, y cuando lo ví me encantó como estaba hecho, por lo que lo utilicé, sin quitar la referencia al autor claro, que aunque no se su

nombre, si esta referenciado el mensaje de donde se cogió en su cabecera.

La utilización de este código es simple, lo declararemos como si fuera una librería en el inicio de nuestro código y a partir de este momento tendremos disponible, varias variables con los datos de las ordenes recibidas desde el PC.

**char cmd\_line** - Cadena de texto con todas las palabras del comando

**char cmd\_str** - Primera palabra de la línea de comandos

**char cmd\_parm\_str** - Array con el resto de las palabras del comando

**int cmd\_n\_parms** – Número de parámetros recibidos

**char cmd\_parm** - Hasta seis parámetros, de doce caracteres cada una, separadas por comas o espacios.

Su manejo es muy simple, tan solo tendremos que llamar a la función process\_command\_line() la cual nos devolverá 1 si a recibido algún comando y 0 si no a recibido nada, si nos devuelve 1 tendremos las variables antes descritas llenas con los datos del comando.

A continuación haremos la función nuestra que comprobara si recibimos los comandos que definamos, a la que llamaremos Get\_cmd() y que llamaremos desde la función loop() para comprobar si recibimos algo.

```
//////////  
//  
// COMANDOS RECIBIDOS DESDE EL PC  
//  
//////////  
void Get_cmd(void)  
{  
    float f;  
    int r;  
    int IndexRele[] = {0,28,32,30,34,36,38,40,42};  
  
    if (process_command_line() == 1)  
    {  
        // CMD  
        if (strcmp(cmd_str, "cmd") == 0)  
        {  
            if (strcmp(cmd_str, "help") == 0)  
            {  
                Serial.println("COMANDOS DISPONIBLES:");  
  
                Serial.println("=====");  
                Serial.println("=====");  
                Serial.println("cmd dimeo [1/2/T] {valor} //Se asigna el {valor} de dimeo a [fase1/fase2/las dos faces]");  
                Serial.println("cmd setrele {index} [on/off] //Se [activa/desactiva] el relé {index}");  
                Serial.println("cmd getrele {index} //Se obtiene el estado del relé {index} retornando [on/off]");  
                Serial.println("cmd ph {valor} //Se manda una orden al sensor de PH");  
                Serial.println("cmd setph {valor} //Se asigna el {valor} al control de PH");  
                Serial.println("cmd reset // Resetea el Arduino");  
                Serial.println("");  
            }  
        // ASIGNA VALOR DE DIMEADO  
        if (strcmp(cmd_parm[0], "dimeo") == 0)  
        {  
            if (strcmp(cmd_parm[1], "1") == 0)  
            {  
                r = (int)atoi(cmd_parm[2]);  
            }  
        }  
    }  
}
```

```

if ((r >= 0) && (r <= 255))
{
Dimeo1 = r;
analogWrite( PIN_P_LUCES1, Dimeo1 );
Pito(40);
}
}
if (strcmp(cmd_parm[1], "2") == 0)
{
r = (int)atoi(cmd_parm[2]);
if ((r >= 0) && (r <= 255))
{
Dimeo2 = r;
analogWrite( PIN_P_LUCES2, Dimeo2 );
Pito(40);
}
}
if (strcmp(cmd_parm[1], "T") == 0)
{
r = (int)atoi(cmd_parm[2]);
if ((r >= 0) && (r <= 255))
{
Dimeo1 = r; Dimeo2 = r;
analogWrite( PIN_P_LUCES1, Dimeo1 );
analogWrite( PIN_P_LUCES2, Dimeo2 );
Pito(40);
}
}
}

// MANEJAR LOS RELES
if (strcmp(cmd_parm[0], "setrele") == 0)
{
r = (int)atoi(cmd_parm[1]);
if ((r >= 1) && (r <= 8))
{
if (strcmp(cmd_parm[2], "on") == 0)
{
SetRele( 9-r, IndexRele[r], HIGH );
Pito(40);
}
if (strcmp(cmd_parm[2], "off") == 0)
{
SetRele( 9-r, IndexRele[r], LOW );
Pito(40);
}
}
}

// LEE ESTADO DE LOS RELES
if (strcmp(cmd_parm[0], "getrele") == 0)
{
r = (int)atoi(cmd_parm[1]);
if ((r >= 1) && (r <= 8))
{

```

```

if (digitalRead(IndexRelay[r])) {
    Serial.print("on");
} else {
    Serial.print("off");
}
}

// Asignar el PH
if (strcmp(cmd_parm[0], "setph") == 0) {
    f = (float)atof(cmd_parm[1]);
    if ((f >= 0.0) && (f <= 14.0))
    {
        CFG.PH = f;
        SetConfig();
        Pito(40);
    }
} else {
    {
        Serial.println("Error: pH rango entre 0.0 a 14.0");
    }
}

// Comando al PH
if (strcmp(cmd_parm[0], "ph") == 0)
{
    ComandoPH(cmd_parm[1]);
    Serial.print("MANDADO = ");
    Serial.println(cmd_parm[1]);
}

// Resetear el Arduino
if (strcmp(cmd_parm[0], "reset") == 0) resetFunc();
}
}
}

```

Esta función realizara varias funciones con los siguientes comandos.

## COMANDOS DISPONIBLES:

---

```

cmd dimeo [1/2/T] {valor} //Se asigna el {valor} de dimeo a [fase1/fase2/las dos faces]
cmd setrele {index} [on/off] //Se [activa/desactiva] el relé {index}
cmd getrele {index} //Se obtiene el estado del relé {index} retornando [on/off]
cmd ph {valor} //Se manda una orden al sensor de PH
cmd setph {valor} //Se asigna el {valor} al control de PH
cmd reset // Resetea el Arduino

```

Para que nuestro sistema sepa que lo que recibe es un comando, todos los comandos tendrán que empezar por la palabra clave “cmd”.

Si queremos cambiar por ejemplo el dimeo de la primera fase de la luz y ponerla a un nivel de 200 mandaremos el comando siguiente

**cmd dimeo 1 200**

Si se lo queremos cambiar a las dos fases a la vez pondremos

**cmd dimeo T 200**

o si queremos activar o desactivar el relé N°3

**cmd setrele 3 on**

**cmd setrele 3 off**

Y si lo que queremos es ajustar el PH como hemos comentado anteriormente simplemente tendremos la sonda de PH y temperatura introducidas en el tarro de tapón correspondiente y en el momento preciso mandaremos la orden pertinente, por ejemplo para el PH4 el siguiente comando.

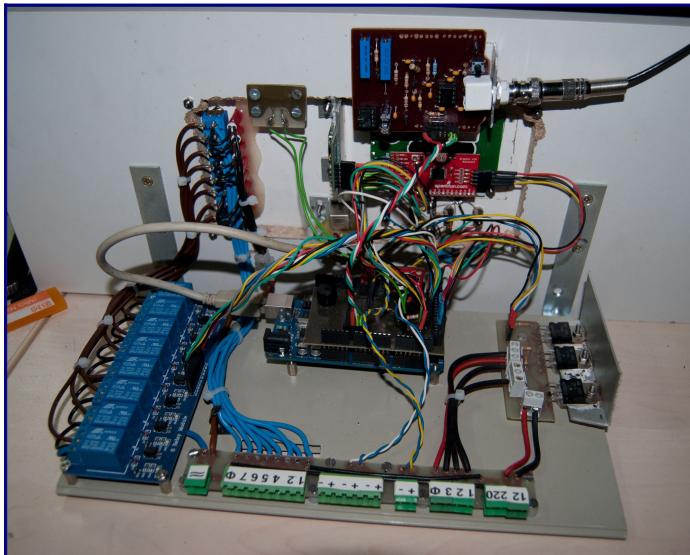
**cmd ph F**

con lo cual el sensor se autocalibrara al valor de PH4.

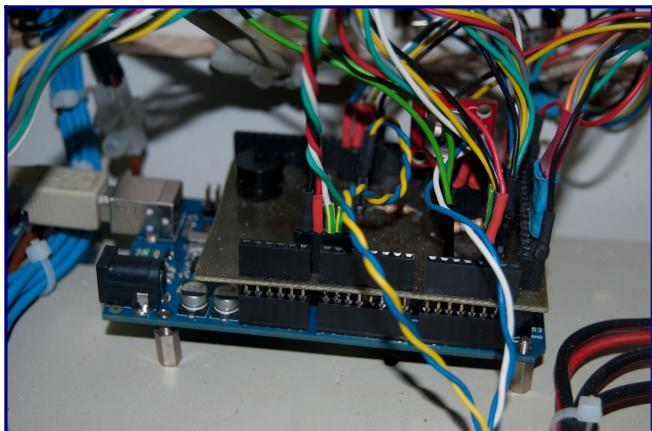
He escrito un tocho bueno, con que espero haberlo echo de una forma clara, de todas formas lo que no comprendáis ya sabéis preguntar.

Aquí os dejo el código de nuestro controlador modificado. [NuestroControlador012.zip](#)

Hoy os voy a poner algunas fotos de mi controlador funcionando, son fotos que le hice antes de montarlo en el mueble del acuario, de el solo cambia con el que tengo actualmente, el sensor de PH, que el que se ve en la foto es uno realizado por mi, que no funcionaba a mi gusto.

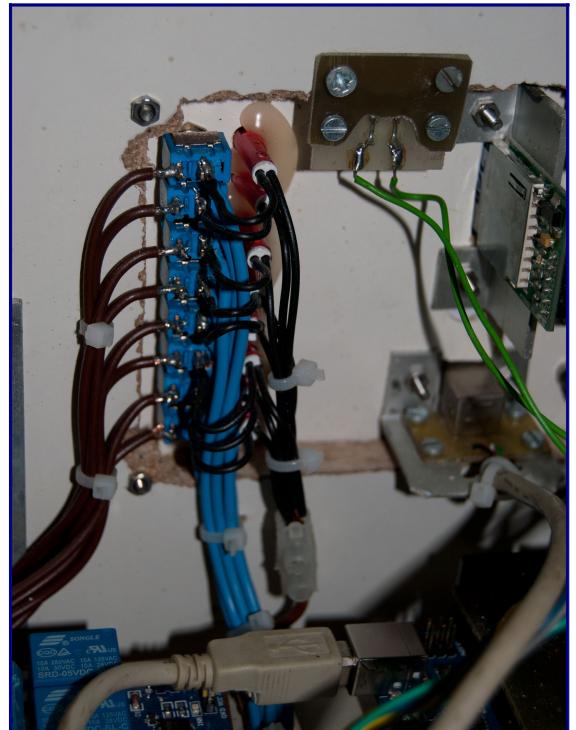


Vista general del controlador.

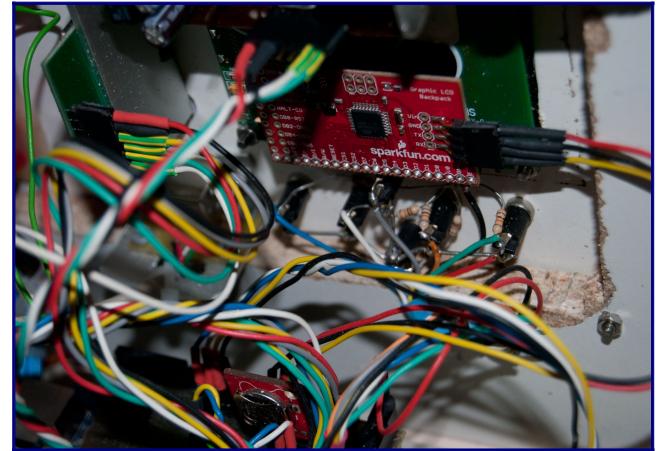
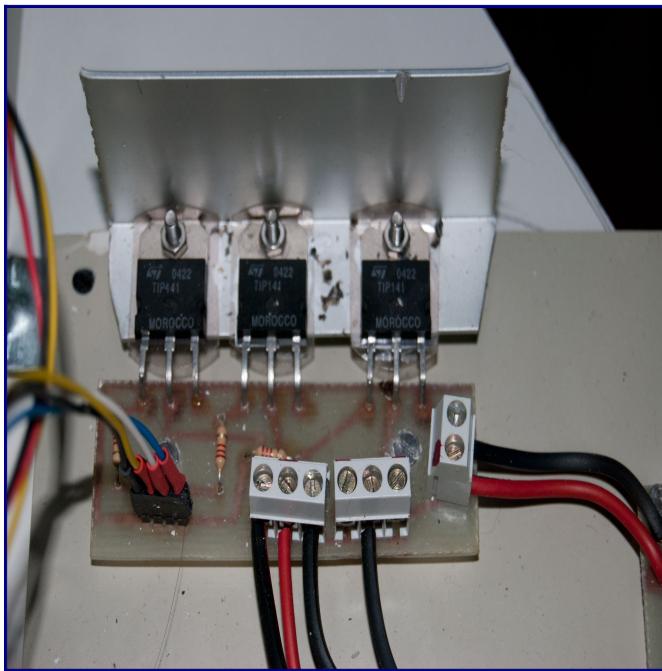


Detalle de la placa de Arduino.

Veréis una tiras de interruptores en el panel, estos interruptores son de seguridad, por si alguna vez el Arduino se empeña en poner algo en marcha y a mi no me conviene, desconectando el interruptor el aparato deja de funcionar aunque el Arduino le ordene funcionar.

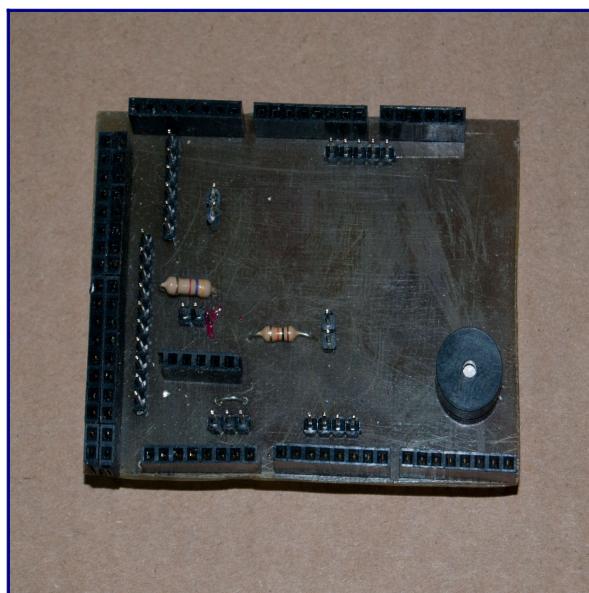


Tira de interruptores.



Detalle de los tip141 para dímejar la pantalla de LEDs.

Tengo que desmontarlo para fotografiar un poco mas de detalle, y el sensor de PH, iré poniendo también esquemas de las placas que e realizado por si las queréis hacer y el código completo de controlador mio.



El panel esta echo con un programa gratuito, el [Front Panel Designer](#). Con el puedes diseñar el panel e imprimirlo en una hoja A4 normal, luego la plastifica en cualquier papelería, ya solo queda pegarla en la caja que vallamos a usar, luego con el panel pegado y seco, se realiza los boquetes y queda bastante bien, yo como lo puse en una tabla y esta era demasiado gorda para los componentes, pegue el frontal plastificado, a una lamina de plástico de 4 mm y luego agarre esta a la madera.

